Analysis of Adaptive Mesh Refinement for IMEX Discontinuous Galerkin Solutions of the Compressible Euler Equations with Application to Atmospheric Simulations

Michal A. Kopera[a,*], Francis X. Giraldo[a]

[a]*Naval Postgraduate School, Department of Applied Mathematics, Monterey, CA 93940*

Abstract

The resolutions of interests in atmospheric simulations require prohibitively large computational resources. Adaptive mesh refinement (AMR) tries to mitigate this problem by putting high resolution in crucial areas of the domain. We investigate the performance of a tree-based AMR algorithm for the high order discontinuous Galerkin method on quadrilateral grids with non-conforming elements. We perform a detailed analysis of the cost of AMR by comparing this to uniform reference simulations of two standard atmospheric test cases: density current and rising thermal bubble. The analysis shows up to 15 times speed-up of the AMR simulations with the cost of mesh adaptation below 1% of the total runtime. We pay particular attention to the implicit-explicit (IMEX) time integration methods and show that the ARK2 method is more robust with respect to dynamically adapting meshes than BDF2. Preliminary analysis of preconditioning reveals that it can be an important factor in the AMR overhead. The compiler optimizations provide significant runtime reduction and positively affect the effectiveness of AMR allowing for speed-ups greater than it would follow from the simple performance model.

Keywords: adaptive mesh refinement, discontinuous Galerkin method, non-conforming mesh, IMEX, compressible Euler equations, atmospheric simulations

1. Introduction

Atmospheric flows are characterized by a vast spectrum of spatial and temporal scales, from weather fronts and planetary waves covering thousands of kilometers and lasting weeks, to turbulent motions at the micro scale. Due to limitations in computational resources, we are not able to resolve all phenomena. Most models assume a uniform mesh and therefore distribute computational resources uniformly across the domain. The scales of motion in the atmosphere are not, however, distributed uniformly both in space and time. The goal of adaptive mesh refinement (AMR) is to focus the resolution of the mesh (and therefore computational resources) where it is most required. Dynamic adaptation aims to follow the important structures of the flow and modify the mesh as the simulation progresses, according to some refinement criterion. An example of such a situation in the atmosphere is a hurricane, which is an event of significance that is relatively localized within a global domain but traverses vast distances. Static adaptation, on the other hand, aims to refine the mesh once at the beginning of the simulation, which allows to focus the computational resources at a particular area of interest in the domain. In such a way one could well resolve a certain part of the globe for which the weather forecast is to be performed, leaving the rest of the domain at much coarser resolution. In this paper we focus on dynamic mesh refinement, which we present on a couple of atmospheric test cases. All the methods that we discuss, however, are readily applicable to static adaptation.

*Corresponding author. Tel: +1 831-656-3247

Email addresses: makopera@nps.edu (Michal A. Kopera), fxgirald@nps.edu (Francis X. Giraldo)

Report Documentation Page

Form Approved
OMB No. 0704-0188

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED
2013		**00-00-2013 to 00-00-2013**

4. TITLE AND SUBTITLE	5a. CONTRACT NUMBER
Analysis of Adaptive Mesh Refinement for IMEX Discontinuous Galerkin Solutions of the Compressible Euler Equations with Application to Atmospheric Simulations	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **Naval Postgraduate School (NPS),Department of Applied Mathematics,Monterey,CA,93943**	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

13. SUPPLEMENTARY NOTES
Journal of Computational Physics (in review 2013)

14. ABSTRACT

The resolutions of interests in atmospheric simulations require prohibitively large computational resources. Adaptive mesh re nement (AMR) tries to mitigate this problem by putting high resolution in crucial areas of the domain. We investigate the performance of a tree-based AMR algorithm for the high order discontinuous Galerkin method on quadrilateral grids with non-conforming elements. We perform a detailed analysis of the cost of AMR by comparing this to uniform reference simulations of two standard atmospheric test cases: density current and rising thermal bubble. The analysis shows up to 15 times speed-up of the AMR simulations with the cost of mesh adaptation below 1% of the total runtime. We pay particular attention to the implicit-explicit (IMEX) time integration methods and show that the ARK2 method is more robust with respect to dynamically adapting meshes than BDF2. Preliminary analysis of preconditioning reveals that it can be an important factor in the AMR overhead. The compiler optimizations provide signi cant runtime reduction and positively a ect the e ectiveness of AMR allowing for speed-ups greater than it would follow from the simple performance model.

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			
unclassified	**unclassified**	**unclassified**	**Same as Report (SAR)**	**34**	

In order to discretize the equations we use the discontinuous Galerkin (DG) method on quadrilateral element grids. The DG method has gained a significant level of interest in recent years and there has been a number of efforts to apply it to hydrostatic [1] and nonhydrostatic [2] atmospheric flows. The method benefits from great data locality and high computation intensity, which helps to scale exceptionally well on a large number of processors. It also supports arbitrary high order, which, among other benefits, allows for accurate handling of the non-conforming edge fluxes, which can arise in the process of mesh adaptation. AMR for DG was successfully applied in different applications (e.g. shock capturing [3, 4], mantle convection [5]). Here we focus on atmospheric flows, particularly the dry dynamics governed by the Euler equations.

The question we would like to answer is: how does AMR benefit a simulation? To try to answer this question we perform a detailed analysis of the cost of AMR comparing this to the results of a uniformly refined simulation. To our knowledge, such a detailed study has not been performed previously. Moreover, we seek to answer this question in light of the use of implicit-explicit (IMEX) time-integrators because for real-world applications (such as nonhydrostatic numerical weather prediction) explicit time-integration is not feasible due to the small time-step restriction of the fast acoustic waves.

The following subsections present a brief overview of the work done on AMR for atmospheric simulation in general and specifically in conjunction with the DG method.

1.1. AMR in atmospheric simulations

Jablonowski [6] and Behrens [7] give a good overview of the state of adaptive atmospheric modeling. The hurricane modeling community was the first to use nested grid techniques for their simulations. In this approach fine scale grids were nested into large-scale coarse meshes and communication was allowed from large to small scales [8, 9] or in both directions [10]. The nested grid can move with the feature it tracks, but often some previous knowledge of the grid movement is required and the number of points remains constant throughout the simulation.

Another example of a mesh modification technique which preserves the number of points and grid connectivity throughout the simulation is mesh stretching, i.e., changing the grid spacing by the use of transformation functions (also known as r-refinement). Examples of such an approach are presented in [11, 12] and more recently in [13].

This paper focuses on an adaptive method which does not involve a moving mesh but rather refines the elements dynamically in regions of particular interest. This technique is typically referred to as dynamic AMR. The first atmospheric models involving AMR were developed by Skamarock and Klemp [14, 15]. They used the technique of Berger and Oliger [16] where a finite difference method is used to integrate the dynamical equations first on a coarse and then on the finer grids. In order to determine the location of finer grids a criterion based on truncation error estimation was used. The technique of Berger and Oliger [16] and Berger and Collela [17] is referred to as block-structured AMR. It was later used in a number of studies including LeVeque [18] and Nikiforakis [19].

The only operational weather model which uses dynamic AMR is OMEGA [20]. OMEGA is based on the finite volume MPDATA scheme, originally developed by Smolarkiewicz [21], which uses unstructured triangular meshes. The dynamic adaptation capabilities were implemented to the MPDATA model by Iselin and coworkers [22]. Another example of AMR which uses triangular elements is the work of Giraldo [23] who used the Lagrange-Galerkin method. In all of these approaches the mesh is obtained by a Delaunay triangulation of the domain given some mesh size criteria. When the criterion indicates that the mesh should be adjusted, the triangulation is performed on the entire domain, and the solution is projected from the old mesh to the new one.

A different approach is presented in the work of Behrens [24, 25], where the triangular elements indicated for refinement are subdivided making sure the conformity of the edges is preserved. The method allows for local refinement without modifying the entire domain. It is well suited for triangular meshes, however the same approach for quadrilateral grids would be difficult as maintaining conformity of locally dynamically refined quadrilaterals is more challenging to achieve. An example of static conforming quadrilateral mesh refinement can be found in [26].

Quadrilateral meshes yield to local refinement easily, provided that we allow non-conforming elements in the grid. Examples of quadrilateral-based dynamic AMR methods for the shallow water equations can

be found in the work of Jablonowski [6]. St-Cyr et al. [27] compare an adaptive cubed-sphere grid spectral element shallow water model with an adaptive finite volume method by [6] to investigate the applicability of tree-based AMR algorithms to atmospheric models.

1.2. Element based Galerkin methods for atmospheric AMR

The high order element based Galerkin methods for AMR in atmospheric applications is a fairly new field of study. These methods present a new set of challenges and possibilities for adaptive mesh refinement. By expanding the solution in a basis of high order polynomials in each element, one can dynamically adjust the order of these basis functions, which can differ across elements. This kind of approach is called p-refinement, and an example of such a technique applied to the shallow water equations can be found in [28].

In the previous section we already mentioned the element mesh refinement (so called h-refinement), which focuses on refining the mesh while keeping the polynomial order constant across the elements. If we choose to allow non-conforming elements, the challenge in this approach is the appropriate treatment of the non-confoming faces. For the DG method one needs to compute the flux across the non-conforming faces. The mathematical solution to this problem was proposed by Kopriva [29] as well as Maday et al. [30] who formulated the mortar method for non-conforming elements. Examples of application of h-refinement to atmospheric flows can be found in [31], where the spectral element method for geophysical and astrophysical applications was used, or [27] where the comparison of a finite volume and spectral element AMR code for the shallow water equations was performed. A recent study by Müller et al. [32] investigates the dynamic adaptation of triangular conforming meshes and addresses the question whether coarsening the mesh in certain areas of the grid affects the solution in a significant way. Brdar et al. [33] perform the comparison of two dynamical cores for numerical weather prediction and mention that the DUNE code, which uses the DG method, has AMR capabilities, however no adaptive mesh examples are discussed in that paper.

The p and h refinement methods can be combined together. [34] shows the application of such an algorithm for the DG method using triangular, non-conforming elements applied to shallow water equations. Another example for this set of equations is presented in [35], where an hp-adaptive DG method on quadrilateral, non-conforming elements for global tsunami simulations is considered.

In this paper we focus on the quadrilateral, non-conforming DG method for the Euler equations and provide an in-depth analysis of the performance of our implementation. To our knowledge this is the first work on tree-based non-conforming AMR for the nonhydrostatic atmospheric equations. Furthermore, to our knowledge no previous work has been published on non-conforming AMR for high-order DG methods for these equations.

This paper is organized as follows: Sec. 2 gives a brief overview of the equations we are solving, Sec. 3 provides an outline of the DG method, Sec. 4 discusses the difference between conforming and non-conforming mesh. In Sec. 5 we describe the details of the mesh adaptation algorithm. Finally, in Sec. 7 we provide the outline of the test cases followed by the discussion of the results in Sec. 8. The paper is concluded in Sec. 9 and supplemented with an Appendix, which describes in detail the formulation of the projection method.

2. Governing equations

Non-hydrostatic atmospheric dynamical processes in NUMA[1] are governed by the compressible Euler equations in conservative form which uses density ρ, momentum $\rho\mathbf{u}$ and density potential temperature $\rho\theta$ as state variables (see, e.g., [36] for other forms). We use the following equation set:

$$\begin{aligned}
&\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \mathbf{u}) = 0, \\
&\frac{\partial \rho \mathbf{u}}{\partial t} + \nabla \cdot (\rho \mathbf{u} \otimes \mathbf{u} + P\mathbf{I}) = -\rho g \mathbf{k} + \nabla \cdot (\mu \rho \nabla \mathbf{u}), \\
&\frac{\partial \rho \theta}{\partial t} + \nabla \cdot (\rho \theta \mathbf{u}) = \nabla \cdot (\mu \rho \nabla \theta),
\end{aligned} \tag{1}$$

[1]NUMA is the name of our model and is an acronym for the Nonhydrostatic Unified Model of the Atmosphere

where $\mathbf{u} = (u, w)^T$ is the velocity field, $\nabla = (\frac{\partial}{\partial x}, \frac{\partial}{\partial z})^T$ is the gradient operator, $\otimes$ is the tensor product, $\mathbf{I}$ is the rank-2 identity matrix, $\mathbf{k} = (0, 1)^T$ is the directional vector that points along the z direction, g is the gravitational acceleration and P is the pressure obtained from the equation of state

$$P = P_0 \left(\frac{R\rho\theta}{P_0} \right)^{\frac{c_p}{c_v}}. \tag{2}$$

The dynamical viscosity μ is varied among the test cases. Note that while this is not the mathematically proper form of the true Navier-Stokes viscous stresses, it is sufficient for the chosen test cases as shown in Giraldo and Restelli [36]. Additional terms requiring definition are the pressure at the lower boundary $P_0 = 1 \times 10^5$ Pa, the gas constant $R = c_p - c_v$ and the specific heats at constant pressure and volume, c_p and c_v.

3. Discontinuous Galerkin method

Giraldo and Restelli [36] describe in detail the discretisation of Eq. (1) for the DG method. Here we outline the weak formulation for the sake of completeness. Note that for the sake of brevity the analysis in this paper was conducted using the weak form only, although both strong and weak forms are implemented in the NUMA software with the non-conforming AMR algorithm.

To describe the DG method, we write Eq. (1) in vector form

$$\frac{\partial \mathbf{q}}{\partial t} + \nabla \cdot \mathbf{F} = S(\mathbf{q}),$$

where $\mathbf{q} = (\rho, \mathbf{U}^T, \Theta)^T$ is the solution vector with $\mathbf{U} = \rho\mathbf{u}$ and $\Theta = \rho\theta$. $\mathbf{F}(\mathbf{q})$ is the flux tensor given by

$$\mathbf{F}(\mathbf{q}) = \begin{pmatrix} \mathbf{U} \\ \frac{\mathbf{U}\otimes\mathbf{U}}{\rho} + P\mathbf{I} - \nabla\left(\mu\rho\nabla\frac{\mathbf{U}}{\rho}\right) \\ \theta\mathbf{U} - \mu\nabla\Theta \end{pmatrix}, \tag{3}$$

and the source term $S(\mathbf{q})$ given by

$$S(\mathbf{q}) = \begin{pmatrix} 0 \\ -\rho g\mathbf{k} \\ 0 \end{pmatrix}. \tag{4}$$

We divide the computational domain Ω into a set of non-overlapping elements Ω_e such that $\Omega = \bigcup_{e=1}^{N_e} \Omega_e$. We define the reference element $I = [-1, 1]^2$ and for each element Ω_e there exists a smooth transformation such that $I = \mathcal{F}_{\Omega_e}(\Omega_e)$. Additionally, if $\Omega_e = \Omega_e(x, y)$ and $I = I(\xi, \eta)$ then $\mathcal{F}_{\Omega_e} : (x, y) \to (\xi, \eta)$ and $\mathcal{F}^{-1}_{\Omega_e} : (\xi, \eta) \to (x, y)$. We employ the notation $\mathbf{x} = (x, y)$ and $\xi = (\xi, \eta)$. The Jacobian of this transformation is given by $\mathbf{J}_{\Omega_e} = \frac{d\mathcal{F}_{\Omega_e}}{d\xi}$ with determinant J_{Ω_e}.

Let ψ_k be a basis function of a space $P_N(I)$ of polynomials of degree N or lower in I where the index k varies from 1 to $K = (N + 1)^2$. The tensor product structure of I allows us to construct such a basis as

$$\psi_k = h_i(\xi)h_j(\eta),$$

where $\{h_i\}_{i=0}^N$ is a basis for $P_N([-1, 1])$ and index k is uniquely associated with the pair (i, j): $k = (i+1) + (N + 1)j$. Let ξ_i be the Legendre-Gauss-Lobatto (LGL) points defined as the roots of $(1 - \xi^2)P'_N(\xi) = 0$, where $P_N(\xi)$ is the N^{th} order Legendre polynomial. The basis functions $h_i(\xi)$ are in fact the Lagrange polynomials associated with LGL points ξ_i. Notice that associated with these points are the Gaussian quadrature weights

$$\omega_i = \frac{2}{N(N + 1)} \left(\frac{1}{P_N(\xi_i)} \right)^2.$$

Let $\mathbf{q}_N$ be the approximation of the solution vector $\mathbf{q}$ on the element Ω_e in the expansion basis ψ:

$$\mathbf{q}_N\left(\mathbf{x},t\right)|_{\Omega_e} = \sum_{k=1}^{K} \psi_k\left(\mathcal{F}_{\Omega_e}^{-1}\left(\mathbf{x}\right)\right)\,\mathbf{q}_k(t), \qquad e=1,\ldots,N_e, \tag{5}$$

where we introduce the grid points $\mathbf{x}_k = \mathcal{F}_{\Omega_e}\left(\left(\xi_i,\eta_j\right)\right)$ and the grid point values $\mathbf{q}_k(t) = \mathbf{q}_N\left(\mathbf{x}_k,t\right)$. The computation of the derivatives of $\mathbf{q}_N$ gives

$$\left.\frac{\partial \mathbf{q}_N}{\partial \mathbf{x}}(\mathbf{x},t)\right|_{\Omega_e} = \sum_{k=1}^{K} \frac{d}{d\mathbf{x}}\left[\psi_k\left(\mathcal{F}_{\Omega_e}^{-1}\left(\mathbf{x}\right)\right)\right]\mathbf{q}_k(t), \tag{6}$$

$$\left.\frac{\partial \mathbf{q}_N}{\partial t}(\mathbf{x},t)\right|_{\Omega_e} = \sum_{k=1}^{K} \psi_k\left(\mathcal{F}_{\Omega_e}^{-1}\left(\mathbf{x}\right)\right)\,\frac{d\mathbf{q}_k}{dt}(t). \tag{7}$$

Concerning the computations of integrals, the expansion defined by Eq. (5) yields

$$\int_{\Omega_e} \mathbf{q}_N(\mathbf{x},t)\,d\mathbf{x} = \int_I \mathbf{q}_N\left(\mathbf{x}\left(\xi\right),t\right) J_{\Omega_e}(\xi)\,d\xi \simeq \sum_{i,j=0}^{N} \omega_i\omega_j\,\mathbf{q}_{k_{ij}}(t) J_{\Omega_e}(\xi_i,\eta_j). \tag{8}$$

With the definitions of the solution expansion and the operations of differentiation and integration in place we can now formulate a DG representation of Eq. (1). Here we consider a nodal formulation with inexact integration as described in [36]. We start with multiplying Eq. (1) by a test function ψ and integrating over an element Ω_e:

$$\int_{\Omega_e} \psi\left(\frac{\partial \mathbf{q}_N^e}{\partial t} + \nabla\cdot\mathbf{F}(\mathbf{q}_N^e)\right)d\Omega = \int_{\Omega_e} \psi\,S(\mathbf{q}_N^e)\,d\Omega_e, \tag{9}$$

where $\mathbf{q}_N^e$ denotes the degrees of freedom collocated in Ω_e. Applying now integration by parts and introducing the numerical flux $\mathbf{F}^*$, the following problem is obtained:
find $\mathbf{q}_N(\cdot,t) \in V_N^{DG}$ such that $\forall\Omega_e$, $e=1,\ldots,N_e$

$$\int_{\Omega_e} \psi\frac{\partial \mathbf{q}_N^e}{\partial t}d\Omega_e + \int_{\Gamma_e} \psi\,\mathbf{n}\cdot\mathbf{F}^*(\mathbf{q}_N)\,d\Gamma_e - \int_{\Omega_e} \nabla\psi\cdot\mathbf{F}(\mathbf{q}_N^e)\,d\Omega_e = \int_{\Omega_e} \psi\,S(\mathbf{q}_N^e)\,d\Omega, \tag{10}$$

$\forall\psi \in L^2(\Omega)$.

The coupling between neighboring elements is then recovered through the numerical flux $\boldsymbol{F}^*$, which is required to be a single valued function on the interelement boundaries and the precise definition of which is given in [36].

By virtue of Eqs. (6), (7) and (8), Eq. (10) can be written in the matrix form

$$\frac{d\boldsymbol{q}^e}{dt} + \left(\widehat{\boldsymbol{M}}^{s,e}\right)^{\mathcal{T}}\boldsymbol{F}^*(\boldsymbol{q}) - \left(\widehat{\boldsymbol{D}}^{e}\right)^{\mathcal{T}}\boldsymbol{F}(\boldsymbol{q}^e) = S(\boldsymbol{q}^e) \tag{11}$$

where $\widehat{\boldsymbol{M}}^{s,e} = \left(M^e\right)^{-1}\boldsymbol{M}^{s,e}$ and $\widehat{\boldsymbol{D}}^{e} = \left(M^e\right)^{-1}\boldsymbol{D}^e$. M^e, $\boldsymbol{M}^{s,e}$ and $\boldsymbol{D}^e$ are local mass, boundary mass and differentiation matrices given by

$$M^e_{hk} = w_h J_{\Omega_e}\left(\boldsymbol{\xi}_h\right)\delta_{hk}, \qquad \boldsymbol{D}^e_{hk} = w_h J_{\Omega_e}\left(\boldsymbol{\xi}_h\right)\boldsymbol{\nabla}\phi_k(\boldsymbol{x}_h), \qquad \boldsymbol{M}^{s,e}_{hk} = w^s_h J^s_{\Omega_e}\left(\boldsymbol{\xi}_h\right)\delta_{hk}\,\boldsymbol{n}(\boldsymbol{x}_h),$$

where $h,k=1,\ldots,K$, δ_{hk} is the Kronecker delta, $\boldsymbol{\xi}_k = (\xi_i,\eta_j)$, $w_k = \omega_i\omega_j$, $w^s_k = \omega_i$ for $j=0$ or $j=N$, $w^s_k = \omega_j$ for $i=0$ or $i=N$ and $J^s_{\Omega_e}$ is the Jacobian of transformation $\mathcal{F}_{\Omega_e}$ restricted to the boundary of I.

Note that this equation can be simplified to yield the following semi-discrete weak form

$$\frac{d\boldsymbol{q}^e_i}{dt} = \left(\widehat{\boldsymbol{D}}^e_{ij}\right)^{\mathcal{T}}\boldsymbol{F}^e_j + S^e_i - \frac{w^{s,e}_i J^s_i}{w^e_i J_i}\left(\boldsymbol{n}^{s,e}_i\right)^{\mathcal{T}}\boldsymbol{F}^*_i. \tag{12}$$

In Eq. (10) we notice that there is only one integral (Γ_e) which couples the neighboring elements together. It is this boundary integral that needs to be modified in order to handle non-conforming AMR. This is explained in detail in Sec. 6.

4. Conforming vs non-conforming mesh

The first question that needs to be answered when constructing the AMR algorithm is whether non-conforming elements are allowed in the grid - that is whether each edge can be shared by more than two elements. We can restrict ourselves to purely conforming ones, where all edges in the mesh are owned by two elements only. In the conforming case, the entire burden of handling the changing mesh falls on the mesh adaptation algorithm, which has to make sure that the grid remains conforming after adaptation. The upside of this approach is an easy communication between neighboring elements. Since their edges are conforming, the AMR does not introduce any additional complication to the DG method solver.

In contrast, in the non-conforming case the mesh adaptation algorithm is kept simple: we divide each element marked for adaptation into a predefined number of children elements. In our case, we choose to divide a 2D quadrilateral element into four children elements (from here on, we shall refer to quadrilaterals as quads). This leads to a situation where, if only one of two neighbor elements is refined, the non-refined neighbor shares an edge with two children elements. This requires the DG solver to be able to compute the numerical flux on such a non-conforming edge. This approach shifts the burden from the mesh adaptation algorithm to the solver side.

In this paper we present the non-conforming approach for a 2D quad-based mesh for the DG method [2]. We believe that the added complication and increased cost related to the computation of fluxes through non-conforming edges is more than made up for by the simplified element refinement algorithm.

5. Mesh adaptation algorithm

5.1. Forest of quad-trees

We adopt the forest of quad-trees approach proposed by [37]. We generate an initial coarse mesh, which has to represent the geometrical features of the domain. In Fig. 1a we present a simple two element initial mesh. We call it the level 0 mesh, where each element is a root for a tree of children elements. If we decide to refine element 1, we replace this element with four children, which belong to the level 1 mesh. We represent it graphically on the right panel of Fig. 1b. Active elements (a set of elements which pave the domain entirely) are marked in blue. Element number 1 is now inactive, replaced by the four newly created elements 3, 4, 5 and 6.

If we further choose to refine element number 5, and thus introduce level 2 elements, we render this element inactive and replace it with the four children 7, 8, 9 and 10. Our element tree is presented in Fig. 1c. Now elements 1 and 5 are no longer active while the active elements 2, 3, 4, 6, 7, 8, 9 and 10 form our mesh.

5.2. Space filling curve

In the previous subsection's examples we assigned numbers to the elements. Those numbers serve as element unique labels. In order to traverse all active elements in the mesh, we utilize the concept of the space filling curve (SFC). To each active element we assign an index, which defines the element's position in the space filling array. In order to index all the active elements in the mesh, we search each quad-tree in the forest for active elements (leaves) starting with the first root element (element 1). Since it is inactive, we move to its first child. If the child element is active, we include it into the space filling curve and move to the next child; if it is inactive we recursively traverse the sub-tree rooted in this element. After finishing the search of one quad-tree, we move to the next level 0 root element.

Figure 2 illustrates the space filling curve concept. Note that the same tree traversing technique can yield different space filling curves, depending on the numbering of children elements. The numbering that produced the curve in Fig. 2a imposes a row-major order of children elements, while the curve in Fig. 2b was generated using counter-clockwise enumeration. The numbering is applied recursively, starting from the

[2]It should be noted that doing this for the continuous Galerkin method is also possible but slightly more complicated. We shall report on this in a follow-up paper.

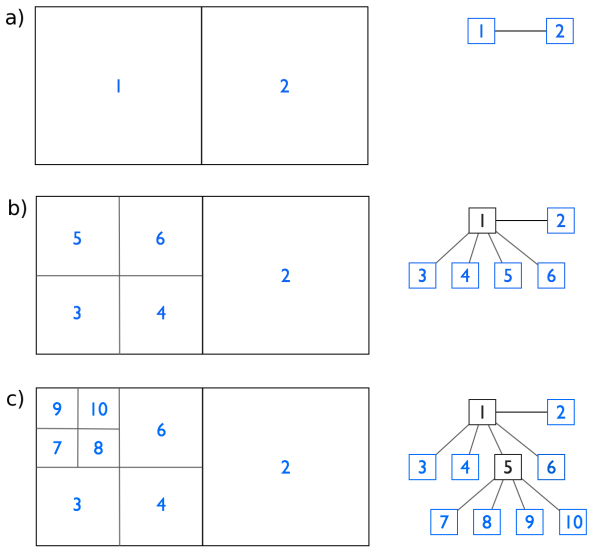

Figure 1: Unstructured grid organized into a forest of quad-trees.

level 0 mesh and traversing the tree to its fullest depth. Therefore in the row-major order we first number the elements of the level 0 mesh as $(1, 2)$. Next we move one level down and enumerate the children of element 1 (element 2 has no children) starting from the bottom left element and enumerating the children in the bottom row first, then move to the second row and enumerate the remaining two elements. We repeat the recursive procedure until we enumerate all the elements on all levels.

In DG methods we prefer indexing the elements in such a way that adjacent elements are placed close to each other in the space filling curve. This increases data locality, which in turn makes the computation of fluxes more efficient. Data locality is particularly important in parallel implementations of the AMR algorithm, however a full study of the influence of element numbering on the efficiency of the code exceeds the scope of this paper.

5.3. Element division technique

To keep the adaptation algorithm simple, and the non-conforming face handling as efficient as possible (see Sec. 6.1), we require that each side of an element to be refined is divided in a 2:1 size ratio. This means that we split a parent edge into two children edges of equal length. In NUMA we use general quadrilaterals, possibly with curved edges, which could prove splitting elements in physical (x, y) space difficult. Therefore we perform the element splitting in the computational space (ξ, η) instead (see Fig. 3a).

In Sec. 3 we defined the transformation $\mathcal{F} : (x, y) \to (\xi, \eta)$. The inverse mapping $\mathcal{F}^{-1} : (\xi, \eta) \to (x, y)$ is simply the expansion of a variable in the polynomial basis ψ:

$$q(x_j, y_j) = \sum_{i=1}^{K} q_i \psi_i(\xi_j, \eta_j),$$

where q is a variable defined in physical space, q_i is the nodal value of the variable in computational space at node i corresponding to a basis function ψ_i. (ξ_j, η_j) represent coordinates of the j-th nodal point in computational space, which corresponds to a point (x_j, y_j) in physical space.

We can treat the x and y coordinates of the nodal points as a variable across the element, which yields

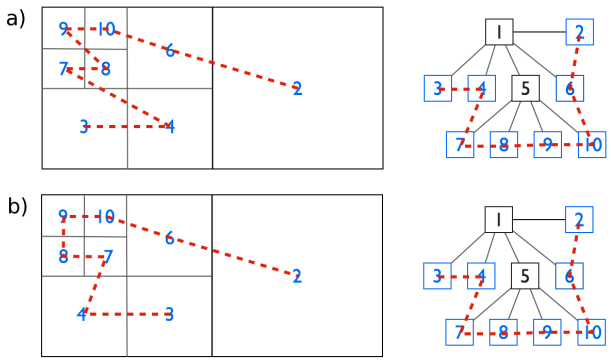

Figure 2: Two different variations of space filling curve.

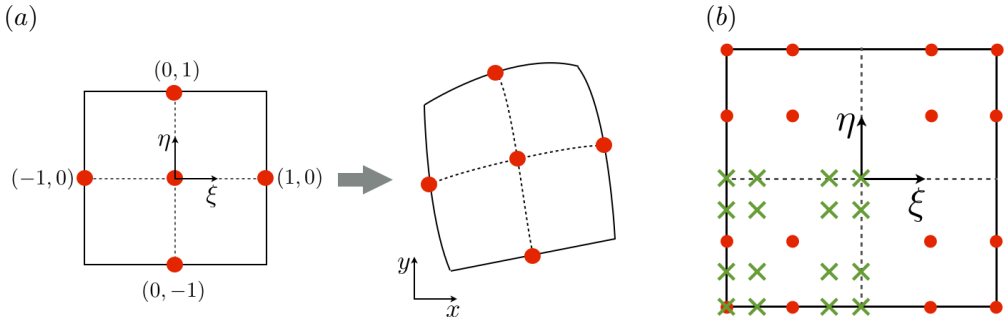

Figure 3: (a) General shape quad division; (b) Projection of coordinates from parent to children - (red) dots represent nodal values of coordinates of parent element, (green) crosses are the nodal coordinates we seek for one of the four children elements.

the expansions:

$$\begin{aligned} x_j \equiv x(\xi_j, \eta_j) &= \sum_{i=1}^{K} x_i \psi_i(\xi_j, \eta_j), \\ y_j \equiv y(\xi_j, \eta_j) &= \sum_{i=1}^{K} y_i \psi_i(\xi_j, \eta_j). \end{aligned} \tag{13}$$

Figure 3b presents an example of a standard element with 16 nodal points (red dots). Each point has a nodal value of x_i^p and y_i^p and is characterized by coordinates (ξ_i^p, η_i^p). If an element is marked for refinement, we split the standard element into four children of equal size (dashed lines). Next, using the parent nodal values of (x_i, y_i) we find the nodal values of (x, y) in the children elements (green crosses in Fig. 3). Since we have (ξ_i^p, η_i^p) coordinates of parent nodal points, then the division of the parent element into four children is very simple. We can easily find (ξ, η) coordinates for children nodal points. Following [29] we write:

$$\begin{aligned} \xi_i^{c(k)} &= s \cdot \xi_i^p - o_\xi^{(k)}, \\ \eta_i^{c(k)} &= s \cdot \eta_i^p - o_\eta^{(k)}, \end{aligned} \tag{14}$$

where $s = 1/2$ is a scale factor and $o_\xi^{(k)}$ and $o_\eta^{(k)}$ are the offsets corresponding to the variables ξ and η respectively for a child k. For the lower left child element (depicted in Fig. 3b) $k = 1$, $o_\xi^{(k)} = 1/2$ and $o_\eta^{(k)} = 1/2$. For different children the offset values will differ in sign.

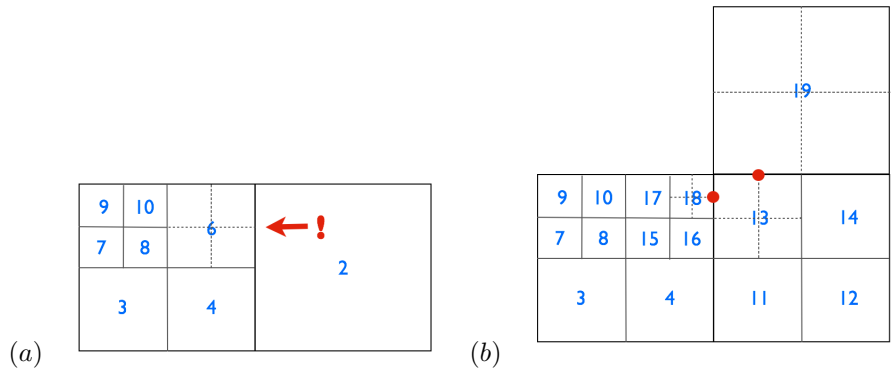

Figure 4: 2:1 balanced mesh and ripple propagation problem.

We seek the values of x and y at child nodal points (green crosses - here only one child is represented). Knowing the value of $(\xi^{c(k)}, \eta^{c(k)})$ at children nodes we can substitute it into the expansion (13) and find the coordinates of the children nodes in the physical space. We write

$$\begin{aligned} x_i^{c(k)} &= \sum_{j=1}^{K} x_j \psi_j(\xi_i^{c(k)}, \eta_i^{c(k)}), \\ y_i^{c(k)} &= \sum_{j=1}^{K} y_j \psi_j(\xi_i^{c(k)}, \eta_i^{c(k)}), \end{aligned} \tag{15}$$

which can be represented in matrix form as

$$\begin{aligned} x_i^{c(k)} &= \mathbf{L}_{ij}^{c(k)} x_j^p, \\ y_i^{c(k)} &= \mathbf{L}_{ij}^{c(k)} y_j^p, \end{aligned} \tag{16}$$

where $\mathbf{L}_{ij} = \psi_j(\xi_i^{ck}, \eta_i^{ck})$ is the interpolation matrix which holds values of the parent basis functions ψ_j at children node coordinates $(\xi_i^{ck}, \eta_i^{ck})$.

5.4. 2:1 balance

A mesh, where each edge is shared by at most three elements, is called a 2:1 balanced mesh, and an edge which is shared by two elements on one side, and one element on the other, is called 2:1 balanced edge. When using the term "2:1 balanced", we assume that a 1:1 balanced edge (where an edge is shared by only one element on each side) is also a valid member of the 2:1 balanced mesh. The element refinement procedure described in Sec. 5.3 may lead to a situation, where an element, which has a 2:1 balanced edge and lies on the refined side of the edge, is marked for refinement, while its neighbor across this edge is not. This would violate the 2:1 balance causing the edge to be shared by a total of four elements (one on one side, and three on the other).

An example of such a situation is shown in Fig. 4a. Elements 3, 4, 7, 8, 9, 10, 6, 2 form a 2:1 balanced mesh. Let us assume that element number 6 was marked for refinement. This would cause a conflict, where the edge shared by elements 2, 4 and 6 is not 2:1 balanced anymore. In order to avoid this situation, a special balancing procedure needs to be introduced.

In the situation presented in Fig. 4a, the solution of the 2:1 balance problem is to refine the element 2 first, before refining 6, even though 2 might not be originally marked for refinement. This would not lead, at any time of this process, to violating the 2:1 balance rule. One might imagine, however, that for a more complex mesh, refining element number 2 might also cause a conflict with other edges owned by 2. Consider a situation depicted in Fig. 4b, where the mesh is initially 2:1 balanced. In this mesh there are

two levels of refinement predefined. Element 19 is a 0th level element, elements 3, 4, 11, 12, 13, 14 are 1st level elements and 7-10 and 15-18 are 2^{nd} level. Let us assume we want to refine element number 18, which would create 3^{rd} level children elements. This would create a conflict with element 13, which is a 1^{st} level element. Therefore we refine element number 13 to the 2^{nd} level, but this creates a conflict with the 0^{th} level element number 19. In order to refine 18, we need to refine element 19 first, then 13, and finally 18 in order to keep 2:1 balance at all times. This causes some regions of the domain to be more refined than required from the refinement criterion - we did not initially intend to refine 13 or 19. Such phenomenon is called the *ripple effect*, where refinement of one element can cause an entire area not directly neighboring the element in question to be refined [38].

It is easy to show that in the 2D case the ripple propagation is limited by the lowest level element in the mesh. In a 2:1 balanced mesh the level difference between neighboring elements can be at most 1. The conflict can occur only when refining an n-th level element which has a neighbor of level $(n-1)$. Therefore we need to bring the $(n-1)$ level element to level n before refining the original element to level $(n+1)$. If in turn the $(n-1)$ level element causes conflict with $(n-2)$ level element, we need to follow the balancing procedure recursively. In the worst case scenario we will propagate the ripple down to 0th level element, which by definition is a root of the element tree. Therefore by refining one n-th level element we may be forced, in the worst case, to refine n other elements. This will cause $4n$ new elements to be created in the areas possibly not indicated by the refinement criterion. $4n$ is typically a very small number since the simulations shown in this work tend to have values $n <= 5$.

In the case of element coarsening, we adopt a different strategy. If coarsening of an element would cause a conflict (consider a situation in Fig. 4b after refining all indicated elements, when we want to de-refine element 19), we do not perform this operation. In order to keep the 2:1 balance we avoid propagating a de-refinement ripple to higher levels. The rationale behind this strategy is that it is better to have more refined elements than we need, rather than lack the resolution in the areas where it is necessary. This way we ensure that we always have the appropriate level of refinement as indicated by the refinement criterion.

5.5. Refinement criterion

Our focus in this paper is the AMR machinery and its particular application to the DG method, therefore we use a very simple mesh refinement criterion. First, we specify the quantity of interest (QOI). It can be either a primitive variable like θ, u, w or ρ, or an expression derived from those variables (i.e. velocity magnitude, absolute value of temperature fluctuation etc.). We then choose a refinement threshold. If the maximum value of the QOI within an element falls below the threshold, the element is marked for refinement. The maximum criterion can be of course replaced by a minimum criterion. It is also worthwhile to consider a gradient, or other derivatives of primitive variables, as a QOI. Throughout this paper we use the potential temperature perturbation as the quantity of interest.

The refinement criterion typically need not be evaluated every time-step, but rather every predefined number of steps, depending on the particular problem. After each evaluation of the criterion for all active elements in the grid, the balancing algorithm is run to eliminate possible conflicts.

6. Handling of non-conforming edges

The previous section described the details of the mesh refinement algorithm. Here we will focus on the implementation of such an algorithm to a DG solver. Sec. 6.1 and 6.2 describe the computation of the flux for the DG method.

6.1. Projection onto 2:1 edges

In the DG method at every time-step we need to evaluate the numerical flux through all the element edges. When allowing non-conforming elements in the mesh, one needs to address the problem of projecting the data between two sides of a non-conforming edge. In our case the non-conformity is limited to 2:1 balanced edges, which makes the data exchange slightly easier than in a general non-conforming case.

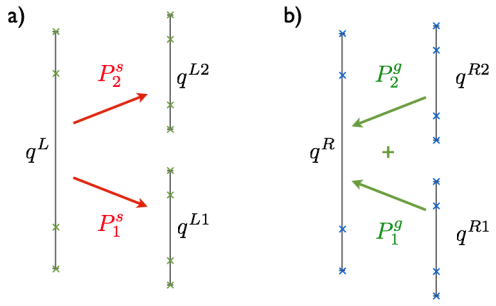

Figure 5: Projection onto non-conforming edges: a) scatter from left parent edge to two children edges and b) gather from two children edges to parent edge.

Consider the situation shown in Fig. 5a. The variable q^L from a parent edge is projected onto two children edges and becomes q^{L1} and q^{L2}. In order to perform this scatter operation we design two projection matrices $\mathbf{P}^{S1}$ and $\mathbf{P}^{S2}$ such that

$$\begin{aligned} q^{L1} &= \mathbf{P}^{S1} q^L \\ q^{L2} &= \mathbf{P}^{S2} q^L. \end{aligned} \tag{17}$$

Similarly for the gather operation we need the matrices $\mathbf{P}^{G1}$ and $\mathbf{P}^{G2}$ which satisfy

$$q^L = \mathbf{P}^{G1} q^{L1} + \mathbf{P}^{G2} q^{L2}$$

The projection matrices are constructed using the integral projection technique [29], derived for different size ratios and different polynomial orders in neighboring elements. In Appendix A we present the outline of the method tailored to 2:1 balanced edges.

Due to the fact that we limit our non-conforming edges to 2:1 ratio only, the scatter and gather projection matrices are the same for all edges and need to be computed only once, which makes the algorithm quite simple and very efficient.

6.2. Flux computation

The flux computation algorithm applied in NUMA to handle non-conforming edges relies on the projection technique described in Sec. 6.1 and Appendix A. First, we scatter the variables from parent to children edges using projection matrices $\mathbf{P}^{S1}$ and $\mathbf{P}^{S2}$. This way we have the necessary information on both sides of children edges just as in the regular conforming case. We compute the numerical flux on children edges using the Rusanov flux (see [36] for details) and gather it back to the parent edge using matrices $\mathbf{P}^{G1}$ and $\mathbf{P}^{G2}$. Finally, the flux is applied on both parent and children edges. Note that you can replace the Rusanov flux with any other Riemann solver.

This algorithm ensures that the amount of numerical flux leaving one element is equal to the flux received by the children elements on the other side of the non-conforming edge. It is worth noting, however, that since we are dealing with the DG method, we allow a discontinuity between variables (and therefore flux) at the children side of the interface. The projection represents the flux from both children elements as one smooth function defined on the parent side of the interface. The parent flux is not point-wise identical to the children fluxes, but we constrain the integral of the flux over the edge, which guarantees conservation.

6.3. 2D projection between parent and children elements

Once an element is refined (derefined), its data must be projected onto its children (parent) elements. In order to perform these two operations we will use the 2D version of the integral projection technique discussed in Sec. 6.1. Figure 6 shows schematically the projections from parent element with coordinates $(\xi, \eta) \in [-1, 1]^2$ to four children, each with separate coordinates $(z_1^{(k)}, z_2^{(k)}) \in [-1, 1]^2$, where $k = 1, \ldots, 4$

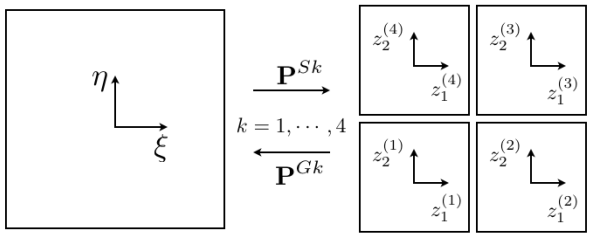

Figure 6: Projection between parent and children elements - 2D extension of integral projection for non-conforming edges.

enumerates the children elements. For this projection we construct the scatter matrix P_{2D}^{Sk}. The inverse operation is performed using the gather matrix P_{2D}^{Gk}. Detailed construction of both matrices is described in Appendix A. Note that while the integral projection method works well for conserved quantities, it may not be appropriate for all variables in the problem. It is sometimes better to interpolate or recompute certain quantities, if possible. An example of such a situation is the gravity direction vector, which is defined completely by the input to the simulation, therefore can be recomputed for each new element in the mesh. In both cases presented in this paper the gravity direction was $\mathbf{k} = (0, 1)$ and in some cases the projection operation caused inconsistencies on the order of the round-off error, which in turn adversely affected the solution.

7. Test cases

In order to test the AMR algorithm we run a selection of cases from the set presented in [36]. The set consists of seven tests widely used for benchmarking of non-hydrostatic dynamical cores of numerical weather prediction codes. For the purpose of benchmarking the AMR capabilities of our code we have picked two scenarios. The density current and rising thermal bubble cases show the performance of the AMR algorithm on a rapidly changing mesh. For both test cases we compare the adaptively refined simulation with a uniformly refined simulation. Both cases are described in detail in the aforementioned paper. Here we outline them for completeness.

7.1. Case 1: Density current

The case was first published in [39] and consists of a bubble of cold air dropped in a neutrally stratified atmosphere. The bubble eventually hits the lower boundary of the domain (no flux wall) and moves horizontally shedding Kelvin-Helmholtz rotors. In order to obtain the grid-converged solution we apply artificial viscosity $\mu = 75m^2/s$(see [39]).

The initial condition is defined in terms of potential temperature

$$\theta' = \begin{cases} 0 & \text{for} \quad r > r_c \\ \frac{\theta_c}{2}\left(1 + cos\left(\frac{\pi r}{r_c}\right)\right) & \text{for} \quad r \leq r_c \end{cases}, \tag{18}$$

where $\theta_c = -15$K, $r = \sqrt{\left(\frac{x-x_c}{x_r}\right)^2 + \left(\frac{z-z_c}{z_r}\right)^2}$ and $r_c = 1$. The domain is defined as $(x, z) \in [0, 25600] \times [0, 6400]$ m with $t \in [0, 900]$ s and the center of the bubble is at $(x_c, z_c) = (0, 3000)$ m with the size of the bubble defined by $(x_r, z_r) = (4000, 2000)$ m. The boundary conditions for all four boundaries are no-flux walls. The velocity field is initially set to zero everywhere.

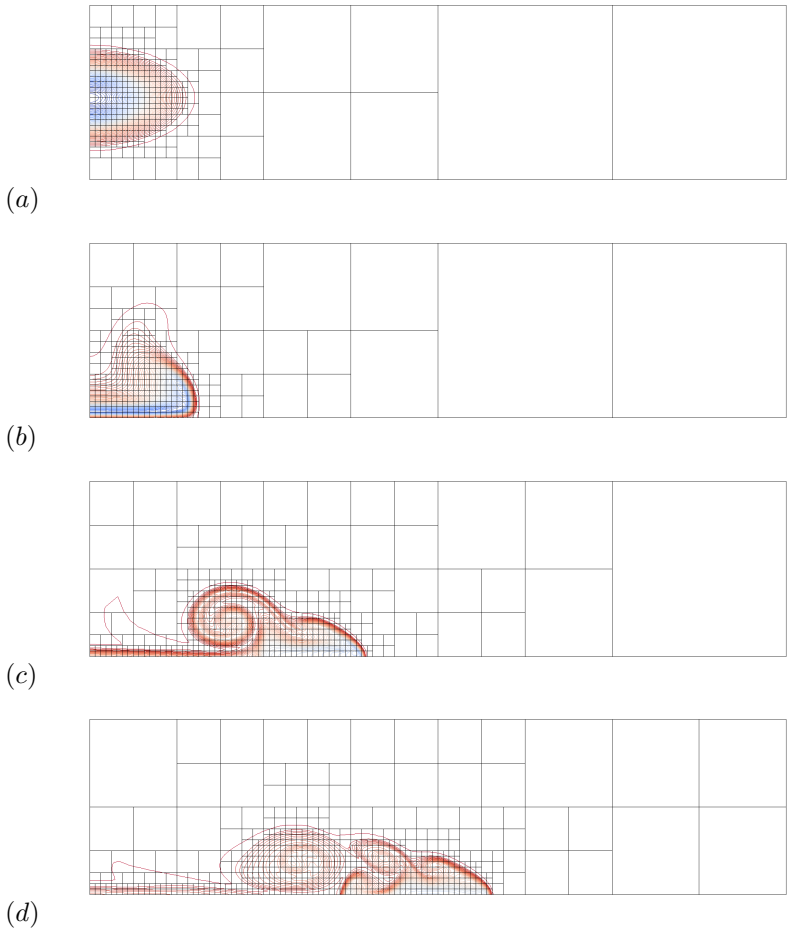

Figure 7: Snapshots of the solution and dynamically adaptive mesh for $\theta_t = 1.0$ at different simulation times: (a) 1s, (b) 300s, (c) 600s, and (d) 900s.

7.2. Case 2: Rising thermal bubble

In this test case a warm bubble rises in a constant potential temperature atmosphere ($\bar{\theta} = 300K$). As it rises, it deforms until it forms a mushroom shape. Initially, the air is at rest and in hydrostatic balance. The initial potential temperature perturbation is given by Eq. (18) with $\theta_c = 0.5$K and $r_c = 250$m. The domain has dimensions $(x, z) \in [0, 1000]$m $\times [0, 1000]$m and the bubble is positioned at $(x_c, z_c) = (500, 350)$m. The boundary conditions for all sides are no-flux. The simulation runs until $t = 700$s.

8. Results

8.1. Case 1 - Density Current

The density current simulation, defined in Sec. 7.1, was initialized with a coarse mesh consisting of four elements (4×1 grid). We chose polynomial order $N = 5$ within each element. The mesh was then refined uniformly to a specified maximum level of refinement. We chose the maximum level to be equal to 5, which allowed for a uniformly fully refined mesh of 128×32 elements. This corresponds to an effective resolution of 40 m, which is slightly below the resolution shown in [36] to yield a converged solution. By effective resolution we mean the average distance between the nodal points within the element. The initial condition was generated for this fully refined mesh, which formed an initial input to all the Case 1 simulations.

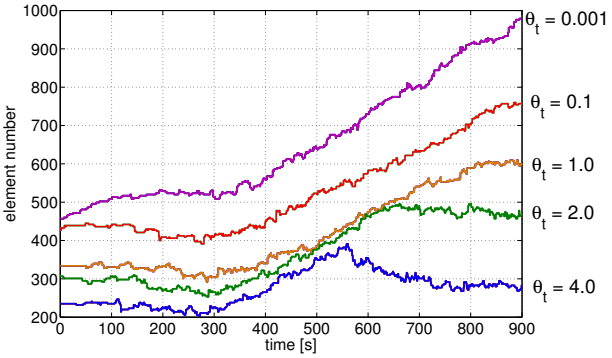

Figure 8: Element count as a function of simulation time for different θ_t threshold values.

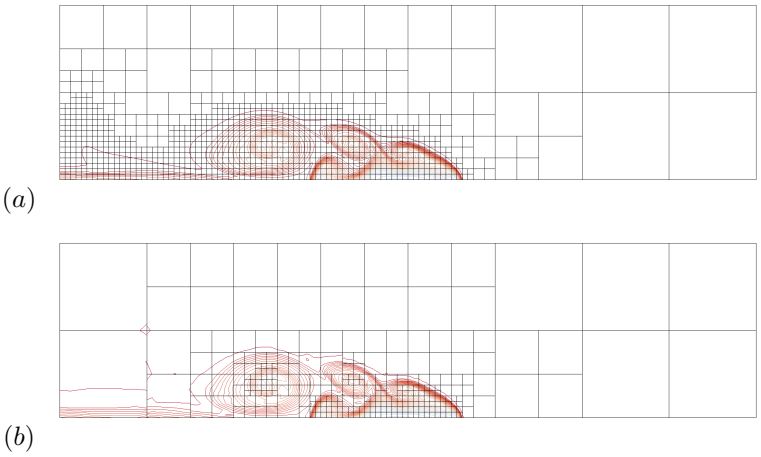

Figure 9: Snapshots of the solution and dynamically adaptive mesh at $T = 900$s for two different thresholds : (a) $\theta_t = 0.001$ and (b) $\theta_t = 4.0$.

In order to obtain a reference solution, we run the fully refined mesh without the AMR algorithm [3]. Next, we select four different refinement thresholds for the potential temperature perturbation $\theta_t = [0.001,\ 0.1,\ 1.0,\ 2.0,\ 4.0]$. For each θ_t the AMR algorithm adapts the mesh to the initial condition and continues modifying the grid every one second of simulation time so that the areas of the domain where $\theta > \theta_t$ are fully refined, and the remaining elements are coarsened to a minimum resolution allowed by the 2:1 balance condition. Figure 7 shows snapshots of the mesh at different times for $\theta_t = 0.1$.

Figure 8 shows the total number of elements in the grid for different values of θ_t over the simulation time. Notice the different behavior of the element count for high and low threshold values. The number of elements for high thresholds tend to level off more quickly as the simulation progresses. Low thresholds cause the element count to steadily rise with time, at least for the time frame considered for this test case. This is due to the fact that the low θ_t criterion not only requires the algorithm to refine the area around the moving structure, but also the near-wall wake, where the temperature is slightly decreased after the passing of the cold front. For further analysis, we define the element ratio (ER = $\frac{N_{ref}}{N_e}$) to be the ratio of the number of elements in the reference simulation ($N_{ref} = 4096$) to the time average number of elements in the AMR simulations (N_e). High ER corresponds to high threshold AMR simulations.

[3]We must use a high-resolution simulation as the reference solution because this test case has no analytic solution.

Table I: Front location (L), minimum potential temperature perturbation (θ_{min}) and simulation runtime (T_s).

θ_t	RK35			BDF2			ARK2		
	L [m]	θ_{min} [K]	T_s [h]	L [m]	θ_{min} [K]	T_s [h]	L [m]	θ_{min} [K]	T_s [h]
ref.	14758.87	-8.90603	60.94	14758.70	-8.90630	6.13	14758.87	-8.90603	9.68
0.001	14758.79	-8.90606	9.78	14758.71	-8.90650	1.08	14758.79	-8.90609	1.57
0.1	14758.81	-8.90607	7.85	14758.74	-8.90661	0.89	14758.81	-8.90610	1.27
1.0	14758.86	-8.90619	6.24	14758.89	-8.90688	0.75	14758.86	-8.90622	1.03
2.0	14758.89	-8.90608	5.53	14758.98	-8.90690	0.71	14758.89	-8.90611	0.92
4.0	14758.80	-8.90555	4.11	14759.10	-8.90648	0.57	14758.80	-8.90558	0.71

8.1.1. Accuracy Analysis

In Fig. 9 two different AMR simulation results are presented. The top picture shows the potential temperature perturbation field for a low threshold ($\theta_t = 0.001$) simulation, while the bottom plot shows a high threshold ($\theta_t = 4.0$) result. The main features of the solution are the same in both cases. Even though in the high threshold simulation the resolved region does not encompass the entire structure, the position of the front and the rotor structure looks identical to the low threshold case. The difference can be noted in the wake of the front and the far field. The high threshold mesh does not capture the wake well, therefore this feature is not represented in the bottom plot.

Table I confirms that all simulations reproduce the main features of the solution well. The simulations were run using an explicit 3^{rd} order 5 step Runge-Kutta method (RK35), an IMEX second-order backward difference formula (BDF2) method and an IMEX second-order Additive Runge-Kutta method (ARK2) - see [40] for details of these time-integrators. The front position was calculated as the location of -1K isotherm at the bottom wall. The values from the computational grid were interpolated using the visualization package Paraview to a very fine (0.01m resolution) uniform grid, and the location of the isotherm was measured on that uniform grid. The front position error does not exceed 1m for all the methods. Interestingly, the front location for RK35 and ARK2 methods are identical. ARK2 also closely follows RK35 with regard to the minimum potential temperature perturbation, which suggests that ARK2 delivers superior accuracy to BDF2. The BDF2 results for both the front location and minimum potential temperature perturbation gives similar, but slightly different results than RK35 and ARK2.

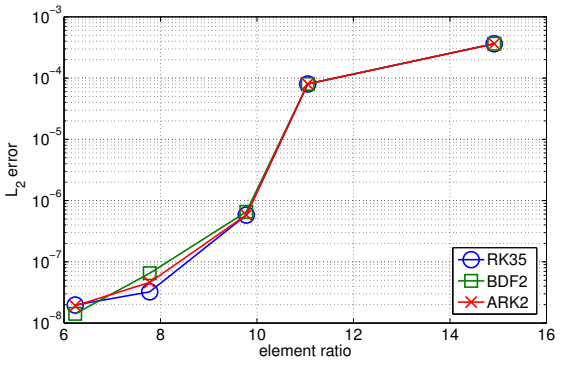

Figure 10: L_2 error norms for AMR simulations using different time integrators: blue line with circles shows RK35 result; green line with squares shows BDF2 result; red line with x markers shows ARK2 result.

Figure 10 shows the L_2 normalized error norms for all AMR simulations plotted against the element ratio. The norm was computed by comparing the potential temperature perturbation field of the AMR simulations with a fully refined reference case with the same time-integration method (AMR explicit compared with reference explicit - blue line with circle markers; AMR BDF2 compared with reference BDF2 - green line with square markers; AMR ARK2 compared with reference ARK2 - red line with cross markers) using the

following formula:

$$L_2(Q,q) = \frac{\sum_{e,k}(Q_k^e - q_k^e)^2}{\sum_{e,k}(Q_k^e)^2}, \tag{19}$$

where Q is the reference solution, q is the AMR solution, e is the index traversing the elements, and k enumerates the nodal points within each element.

The error for all the time integration methods is the same, which shows that AMR equally impacts the simulation accuracy regardless of the time integration scheme. For low ER simulations the L_2 error is very small and stays below 10^{-6}. For high ER the error grows to 10^{-4}. This indicates that for low ER cases the entire domain is adequately resolved, while for high ER simulations the unresolved far field impacts the global accuracy of the solution.

Overall, the error analysis shows that AMR can deliver an accurate result, with the level of accuracy dependent on the refinement threshold. Even the high threshold AMR simulations can represent the main features of the solution well. Regarding IMEX methods, ARK2 seems to deliver more accurate solutions (i.e. the solution which is closer to the explicit RK35).

8.1.2. Performance Analysis

We examine the performance of three time integration methods in conjunction with an adaptive mesh: the explicit RK35, IMEX BDF2, and IMEX ARK2 methods. While the explicit method proves to be simpler to analyze, IMEX is the method of choice for all real-world applications, because it relaxes the explicit time step constraint. For explicit simulations the time step ($\Delta t = 0.01$s) was ten times smaller than for the IMEX simulations. Figure 11d reflects that difference, as the IMEX simulations run nearly 10 times faster than their explicit counterparts. The fastest method is the BDF2, while the ARK2 is just slightly slower running with the same time step (although the ARK2 method can use a larger time-step than BDF2 due to a larger stability region). The Courant number for the IMEX simulations was 1.6.

Since the AMR simulations have a lower element count, we expect them to run proportionally faster than a fully refined reference simulation. The theoretical ideal speed-up is equal to the ratio of the number of elements in the reference case to an average number of elements in the AMR simulations. We expect that the AMR algorithm incurs an overhead for the evaluation of the refinement criterion, projection of the solution between dynamically changing meshes and computation of non-conforming fluxes, therefore the actual speed-up curve should never exceed the ideal profile. The difference between the ideal and actual speed-up curves is a measure of the AMR overhead.

Figure 11 shows the speed-up of the AMR algorithm for different ER values for (a) explicit RK35, (b) IMEX BDF2 and (c) all three time integration methods. The black solid line without markers represents the ideal theoretical speed-up. The speed-up of explicit AMR simulations (blue solid line with circular markers in Fig. 11a) is nearly ideal, which indicates that the cost of performing all the operations accredited to AMR is negligible compared to the cost of time integration. Since in this case the refinement criterion was evaluated every second (i.e., every 100 time-steps) we plot the speed-up of the simulation with refinement check every time-step in the blue dashed line. The overhead due to frequent criterion evaluation shows on the plot, but is still very small. This also indicates that the leading cost in the AMR overhead is the evaluation of the criterion, since when we take away the majority of that cost (solid blue line with circular markers) the speed-up curve overlaps with the ideal line. The other components of the overhead (mesh manipulations, data projections, non-conforming flux computations) have a negligible cost.

In order to validate the choice of refinement frequency, we plot in Fig. 12 the time history of the element count for two extreme refinement thresholds using refinement every time-step (black lines) and every one second (100 time steps for explicit, or 10 time steps for IMEX simulations). Both curves overlap - only small differences are visible for the high threshold simulation near time $T = 680$s, which can be attributed to "blinking", that is refining and de-refining the same mesh location every time step. In this case the less frequent refinement works to our advantage by removing this unwelcome feature.

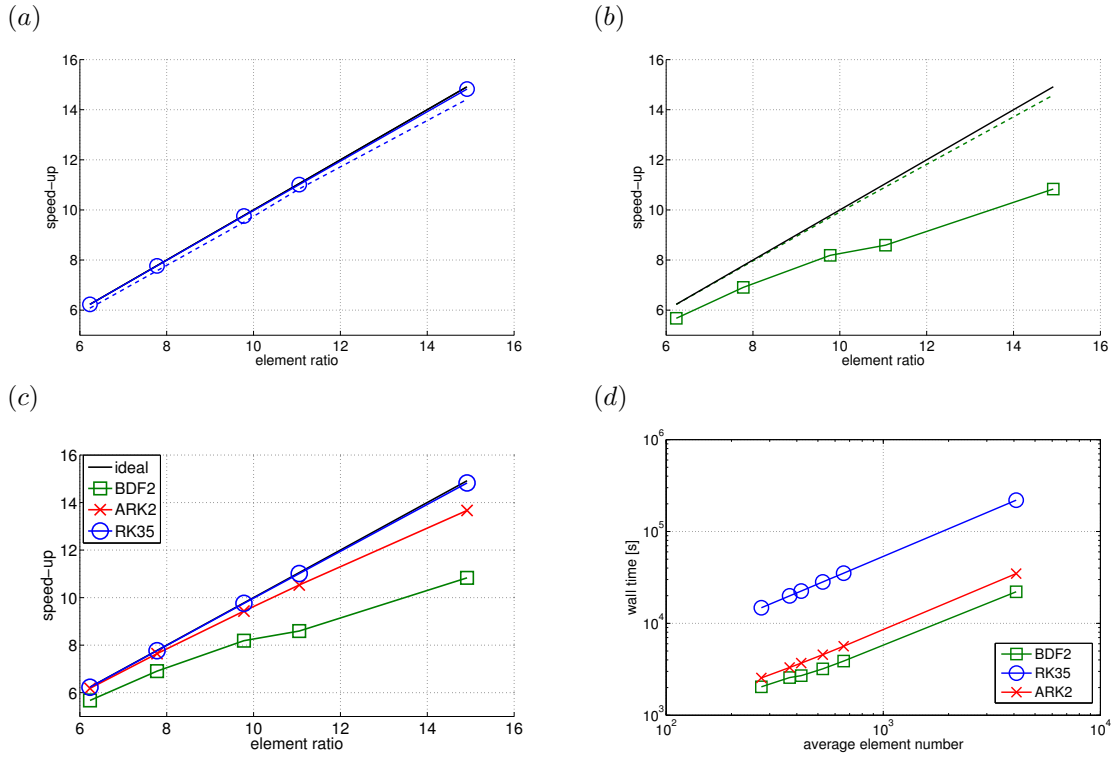

Figure 11: Speed-up for (a) RK35, (b) BDF2 and (c) all time-integrators. Black solid line without markers indicates ideal speed-up. Blue line with circles marks the RK35 results, green line with squares marks BDF2 and red line with crosses marks ARK2. (a) Dashed line shows the speed-up in the case where the refinement criterion is evaluated at every iteration. (b) The dashed line shows the speed-up with a constant number of solver iterations. (d) Wall clock time for all time integration methods.

In Fig. 11b the green solid line with square markers represents simulations with IMEX BDF2 time integration. Clearly the overhead incurred by adaptive simulations is much greater than in the explicit case. The speed-up curve has a variable slope, which indicates that the overhead changes with the choice of refinement threshold. The speed-up of AMR simulations with the IMEX ARK2 method (red line with cross markers in Fig. 11c) is much better than the BDF2, but with bigger overhead than the RK35. Also the slope for ARK2 seems to be less variable than for BDF2.

At the heart of our IMEX methods is the GMRES iterative solver. In Fig. 13 we show the average number of GMRES iterations per time step as a function of ER for both BDF2 (green line with square markers) and ARK2 (red line with cross markers). The point corresponding to ER $= 1$ is the reference simulation. For BDF2 the average iteration count grows with ER, while for ARK2 it remains more or less constant. This can explain the difference in the overhead incurred by AMR with those two time integration methods. To investigate the matter further we run the BDF2 simulations with a prescribed number of GMRES iterations. The result of this exercise is depicted by the dashed line in Fig. 11b. The nearly ideal speed-up is regained by introducing a constant number of GMRES iterations for each simulation. Of course such a constraint is artificial, as the GMRES algorithm automatically determines the number of iterations needed to satisfy the solution accuracy. This means, however, that it is indeed the variable GMRES iteration count that is preventing the AMR BDF2 simulations from achieving a good speed-up.

To investigate the reason for the higher average number of iterations per time step in the AMR BDF2 simulations we plot the time history of the iteration count for both the reference and $\theta_t = 0.001$ simulations in Fig. 14a. The top plot shows the GMRES iterations for the reference simulation - the number oscillates between 7 and 8 every time step. The bottom plot represents the AMR simulation and reveals frequent

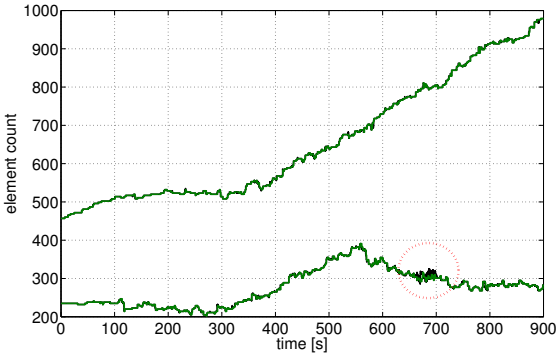

Figure 12: Element number time history for two different refinement threshold settings ($\theta_t = 0.001$ top line; $\theta_t = 4.0$ bottom line) evaluating the criterion every time step (black) and every 100 time steps (green) with RK35.

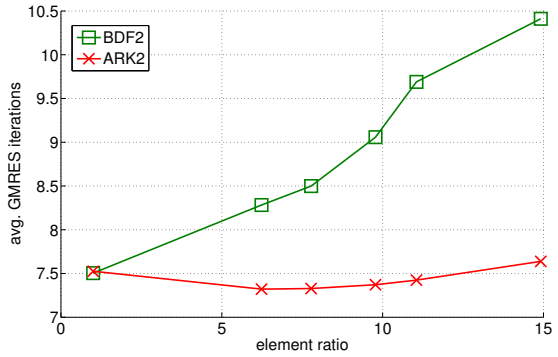

Figure 13: Average number of GMRES iteration per time step for reference and adaptive simulations.

spikes in the iteration count. A closer look at the iteration count compared with the changes in the number of elements in the mesh (Fig. 14b) shows that the spike in the iteration count occurs whenever there is a change in the mesh. This clearly implies that the AMR algorithm incurs an additional overhead with the IMEX BDF2 method because of an increased number of GMRES iterations.

On the other hand, such spikes do not occur for ARK2. The iteration count history in Fig. 14c is very similar for both the reference and $\theta_t = 0.001$ simulations, however much more variable compared with the BDF2 reference. The reason for the different behavior is that even though both BDF2 and ARK2 use the same iterative solver, the system solved by GMRES actually differs between the two methods. While not exhaustive, this result does seem to show that BDF2 is less robust than ARK2 with respect to the projections of the solution between meshes that is introduced by the AMR algorithm. One simple explanation is that BDF2 is a multi-step method (requires the solution at two time-levels in addition to the right-hand-side vector) while ARK2 is a single-step multi-stage method (which only requires the solution at one previous time level and all stages are built directly from this).

8.1.3. Mass conservation

An important measure of the quality of the discretization is the mass conservation. In order to show that our non-conforming AMR implementation does conserve mass as well as a conforming DG method, we investigate the mass conservation error. We define the mass conservation error as:

$$M(t) = \frac{m(0) - m(t)}{m(0)}, \tag{20}$$

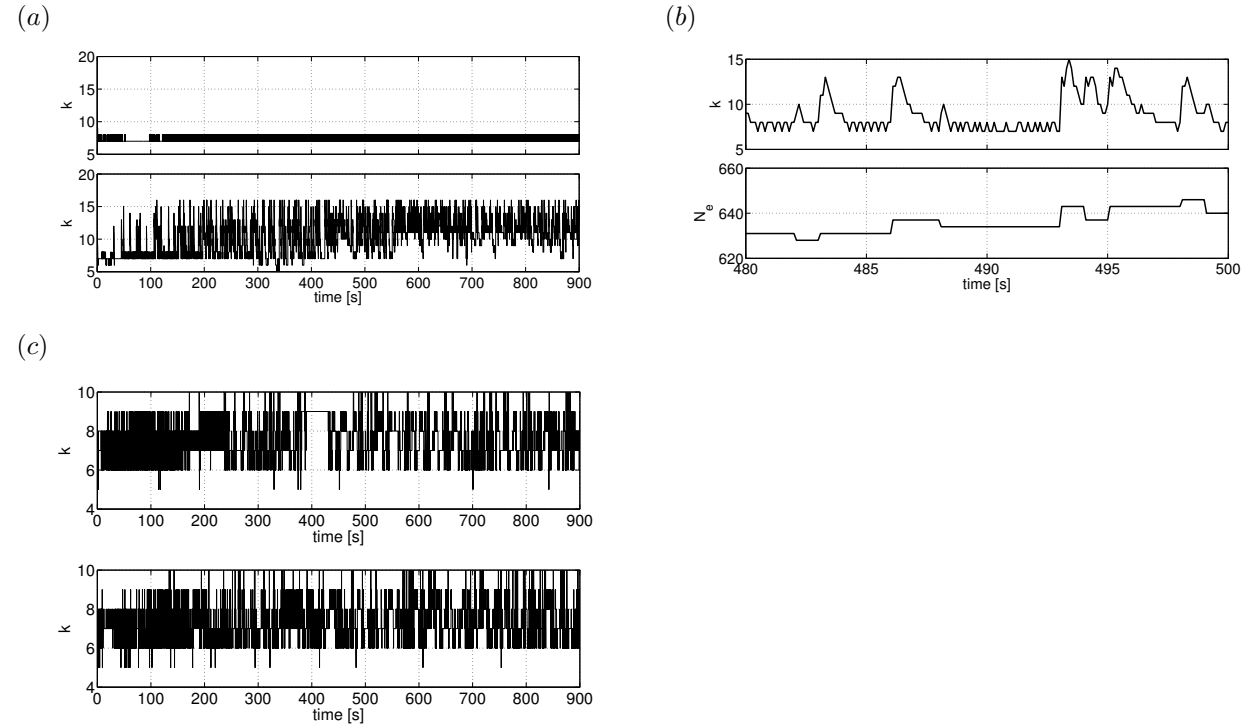

Figure 14: Time history of GMRES iterations (k) for reference (top figure on each panel) and $\theta_t = 0.001$ (bottom figure on each panel) simulations for (a) BDF2 and (c) ARK2 simulations. Panel (b) contains the close-up of the time history of k between $t = 480$s and 500s for the BDF2 simulation (top figure) and the time history of the element count (N_e on the bottom figure).

where $m(t) = \int_\Omega \rho(t) d\Omega$ is the total mass in the system at a time t. The mass conservation error is therefore a normalized measure of the mass loss in the system over the simulation time. The plots of M for the reference simulation and two different AMR simulations are presented in Figure 15. The top three panels present the mass conservation error as a function of time for the reference simulation (Fig. 15a) and two AMR simulations with different refinement thresholds (Fig. 15b and 15c). The M profile of the reference simulation, after the initial transient behavior, is flat for all the time integration methods and stays below 10^{-13}. The profiles for both AMR simulations are bounded from above by the reference mass conservation level. We notice also that M varies slightly over time for the AMR simulations. In panel (d) we plot the element count (N_e) for the reference simulation and two AMR simulations. Note that the level of mass conservation on panels (a), (b) and (c) is correlated with the element count for reference, $\theta_t = 0.001$ and $\theta_t = 4.0$ simulations. The mass conservation error and the element count are both constant for the reference simulation. For the AMR simulations M is growing whenever there is an increase in N_e. This can be explained by the fact that more integration points mean more roundoff errors in mass computations, which add up to create a larger mass conservation error although it remains far below the conservation error of the high-resolution uniform simulation.

8.1.4. Optimization remarks

The NUMA code is written in the Fortran90 language, which relies heavily on the compiler. All presented analysis was conducted with compiler optimization turned off in order to investigate the algorithm itself. Figure 16a presents the decrease in runtime due to using O3 (Intel) compiler optimization setting for both explicit and IMEX simulations. Use of optimization provides almost an order of magnitude faster code, which is a considerable benefit for any application. Figure 16b shows the speed-up analysis for the adaptive simulations using compiler optimization. Both explicit and IMEX speed-up curves (dashed lines) are above

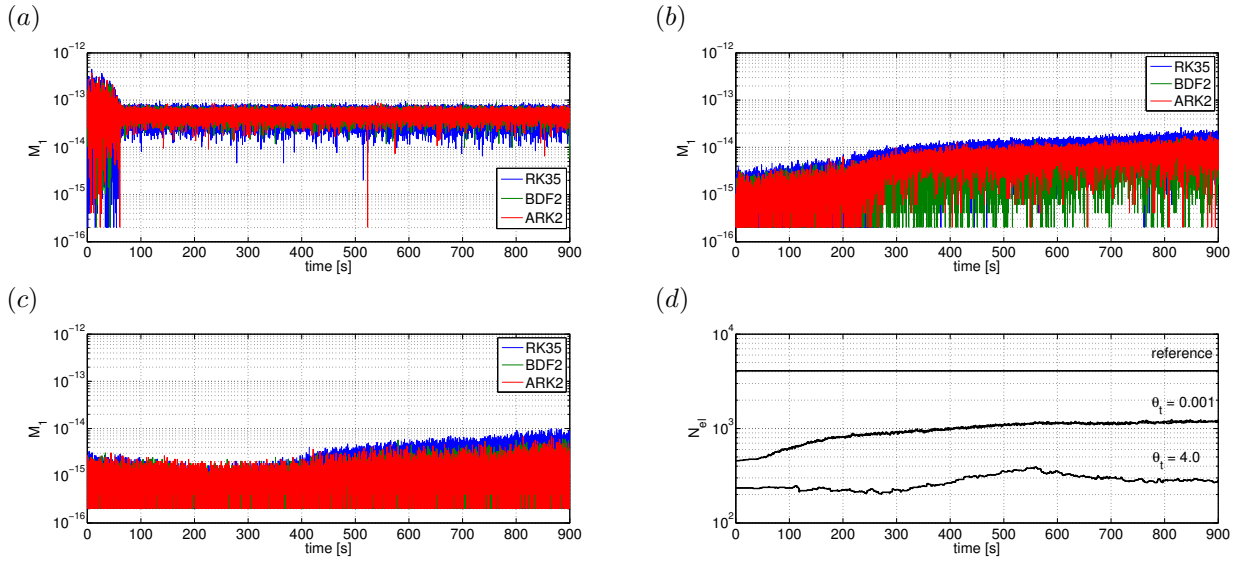

Figure 15: Mass conservation error for Case 1 for (a) reference simulation, (b) $\theta_t = 0.001$ AMR simulation and (c) $\theta_t = 4.0$ AMR simulation for three time integration methods. Panel (d) shows the time history of the element count for all three simulations using all three methods. Results from different simulations with the same time-integration method overlap, which is expected.

Table II: Timings (in seconds of runtime) and percentage breakdown of different AMR components for Case 1.

	RK35		BDF2		ARK2	
	$\theta_t = 0.001$	$\theta_t = 4.0$	$\theta_t = 0.001$	$\theta_t = 4.0$	$\theta_t = 0.001$	$\theta_t = 4.0$
total	5292 (100%)	2128 (100%)	643.6 (100%)	297.0 (100%)	967.8 (100%)	403.1 (100%)
volume integrals	2876.1 (54.3%)	1183.3 (55.6%)	274.46 (42.6%)	135.2 (45.5%)	362.9 (37.5%)	153.7 (38.1%)
face integrals	507.67 (9.59%)	182.19 (8.56%)	66.59 (10.3%)	28.07 (9.45 %)	80.1 (8.29%)	29.58 (7.34%)
non-conforming faces	116.79 (2.21%)	90.94 (4.27%)	21.33 (3.31%)	20.98 (7.06%)	23.65 (2.44%)	20.14 (4.99%)
AMR:	6.030 (0.11%)	2.933 (0.14%)	6.291 (0.98%)	3.154 (1.06%)	6.025 (0.62%)	3.037 (0.75%)
criterion evaluation	3.463	1.422	3.517	1.468	3.527	1.506
mesh manipulation	0.051	0.049	0.05	0.057	0.051	0.063
data projection	0.965	0.581	1.247	0.77	0.965	0.599
other	1.412	0.787	1.450	0.833	1.454	0.843

the theoretical ideal speed-up line. The IMEX time integrators benefitted more from optimization, as they provide higher speed-ups than the explicit method (even though it still suffers from a variable number of GMRES iterations in the BDF2 case). The biggest beneficiary (speed-up wise) of the compiler optimization is the ARK2 method. Also, the slope of all optimized curves is higher than the ideal slope. This indicates that the original assumption about the work load being proportional to the number of elements does not necessarily hold. The theoretical performance model of the algorithm does not incorporate possible optimizations that occur at the compiler level. We attribute those super-speed-ups to the optimization of memory accesses (e.g., prefetching, etc.), which plays a significant role when the problem size gets smaller (and fits in cache) due to the use of the AMR algorithm.

8.1.5. AMR cost breakdown

In Table II we present the absolute runtime (in seconds) and a percentage share in the total simulation time of selected parts of the code for three time integration methods and two refinement thresholds. We list the total runtime, the cost of evaluating the volume integrals (the Ω_e integrals in Eq. (10) apart from the one containing the time derivative), conforming and non-conforming face integrals (both contribute to the Γ_e integral in Eq. (10)) and the time spent in the AMR subroutine. We further provide a breakdown of how much time was spent doing particular AMR tasks like the criterion evaluation, mesh manipulation

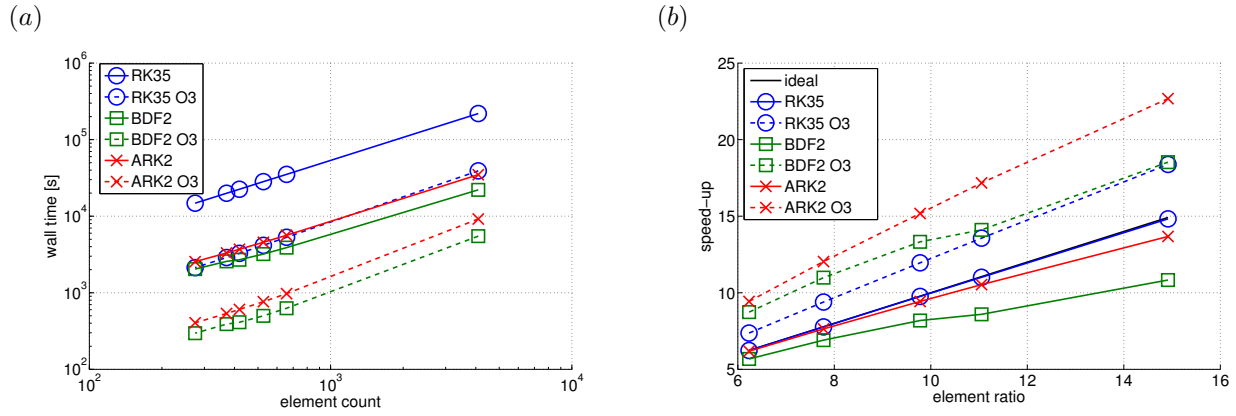

Figure 16: (a) Wall clock time for explicit (blue circles), IMEX BDF2 (green squares) and IMEX ARK2 (red crosses) simulations with (dashed line) and without (solid line) compiler optimization; (b) Speed-up of explicit (blue circles), IMEX BDF2 (green squares) and IMEX ARK2 (red crosses) AMR simulations with (dashed lines) and without (solid lines) compiler optimization.

or data projection. The category "other" contains the operations like recomputing quantities such as the mass matrix, gravity vector etc. which were not optimized for a changing mesh. Instead of recomputing those values for new elements only, we do it for all the elements in the new mesh. The overall time spent in those subroutines is negligible compared to the total simulation time, therefore the optimization would yield a very small gain. It would have to be addressed though in the future.

We observe that the total share of the AMR in runtime is on the order of 1% for IMEX methods and a factor of 10 less for the explicit time integration. All the timings for the AMR part are basically identical for all three methods, which is expected as we call the AMR subroutine every 1s of the simulation time, regardless of the time-step. We also note that the biggest part of the AMR time share is spent in evaluating the refinement criterion, with the data projection cost being significantly smaller and the mesh manipulation negligible. We disregard the cost of other operations, as explained above.

For the sake of argument one can say that if we checked the refinement criterion every time step, the cost of AMR would rise from 1% to 10%. That would be true if the simulation required mesh modifications every time step. As shown in Fig. 12, this is not the case here, as even with refinement check every 10 (or 100 for RK35) time-steps we reproduce the same mesh behavior as for refinement every time-step. Even then, the 10% cost is still an acceptable one.

The evaluation of the flux over the non-conforming faces can be also considered an overhead of AMR, however it is more difficult to quantify how much of the cost can be actually attributed to AMR. As explained in Sec. 6, the cost of evaluating the non-conforming flux does not only include computing the Rusanov flux on two children faces, as in a regular conforming case, but also the projections from and to the parent face. The cost of the operations on the non-conforming face is then larger than the cost for the two conforming faces replaced by it. On the other hand, the non conforming face allows for decreasing the number of elements in the domain, and therefore limits the number of expensive volume integral evaluations.

8.2. Case 2 - Rising thermal bubble

In a similar fashion to Case 1, we start the rising thermal bubble simulation by refining the level 0 mesh of 2×2 elements uniformly by 5 levels. On the resulting mesh of 64×64 elements we generate the initial condition described in Sec. 7.2. The polynomial order is again set to $N_P = 5$, which gives the effective resolution of 3.125m. The smaller domain size and increased resolution, compared to Case 1, imposes a decreased time step. We choose $\Delta t = 0.025$s, which results in the Courant number of 4.7. This set-up generates a problem of a similar size to Case 1, but with significantly increased runtime due to the smaller time-step. Also, an increased Courant number will put more effort on the GMRES solver, as the iteration count will be significantly larger.

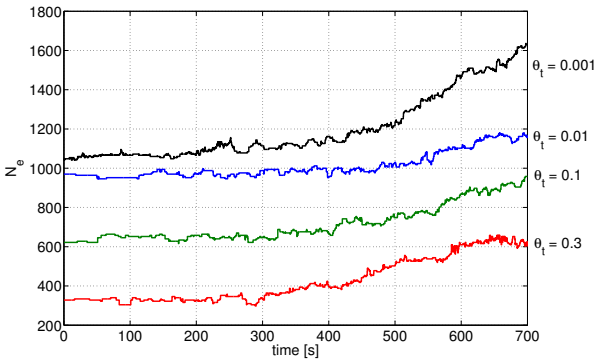

Figure 17: History of element count for different refinement threshold settings for Case 2.

Table III: Maximum potential temperature perturbation (θ_{max}), height of the bubble (H_b) and simulation runtime (T_s).

θ_t	RK35			BDF2			ARK2		
	H_b [m]	θ_{max} [K]	T_s [h]	H_b [m]	θ_{max} [K]	T_s [h]	H_b [m]	θ_{max} [K]	T_s [h]
ref.	961.09	0.46110	197.51	960.16	0.46124	91.29	961.27	0.46064	77.38
0.001	961.07	0.46110	58.51	960.75	0.46012	28.50	961.74	0.45993	23.64
0.01	961.05	0.46116	49.67	960.88	0.46031	25.79	961.81	0.46001	19.99
0.1	962.27	0.46049	35.48	962.15	0.45861	19.43	963.02	0.45853	14.98
0.35	968.92	0.45795	18.13	967.76	0.45551	10.22	968.34	0.45759	7.95

Figure 17 shows the time history of the element count for four different refinement thresholds. The behavior of the mesh is similar to Case 1; the mesh initially does not change much. The number of elements starts growing at later times. For high thresholds this growth is subsequently stopped ($\theta_t = 0.3$ for $t > 600$s). The period of the initial inactivity is longer by 100s than for Case 1, and has a much larger share in the total simulation time (over 50%).

8.2.1. Accuracy analysis

Figure 18 compares four potential temperature perturbation fields at $t = 700$s for different refinement thresholds. Figure 18a shows that the lowest threshold refines a large portion of the domain around the bubble, including the interior of the mushroom bubble and the wake. This results in very smooth contour lines. As the threshold rises, a smaller portion of the domain is refined and the contours become more wavy indicating some instability. Note that in Figs. 18b and 18c we see that the outer contours of the mushroom are refined in the same way, but the mesh within the bubble is very different, which results in a more wavy mushroom pattern in Fig. 18c. This indicates that not only the strong gradient zone, but also the internal mushroom region is important for this case. The solution in Fig. 18d is the worst case scenario where the general bubble shape is still recreated, but the solution is not smooth and some mesh imprinting can be seen at the interfaces of big elements in the center of the mushroom. In this case only the highest potential temperature perturbation areas are refined and the lack of refinement in the rest of the bubble affects the solution in the refined region significantly.

Table III summarizes the maximum potential temperature perturbation in the domain at time $T = 700$s, as well as the position (height) of the bubble at that time. The position of the bubble is defined as a vertical coordinate of the top-most cross-section of $\theta = 0.1$K isoline and $x = 500$m central axis of the bubble. Additionally the runtime for all the cases using different time integration methods is presented for comparison. The runtime was measured for simulations without compiler optimizations and preconditioning.

For all time integration methods, both the bubble height and θ_{max} follow the reference solution closely for low θ_t and diverge for larger values of the refinement threshold. The simulation with $\theta_t = 0.01$ matches

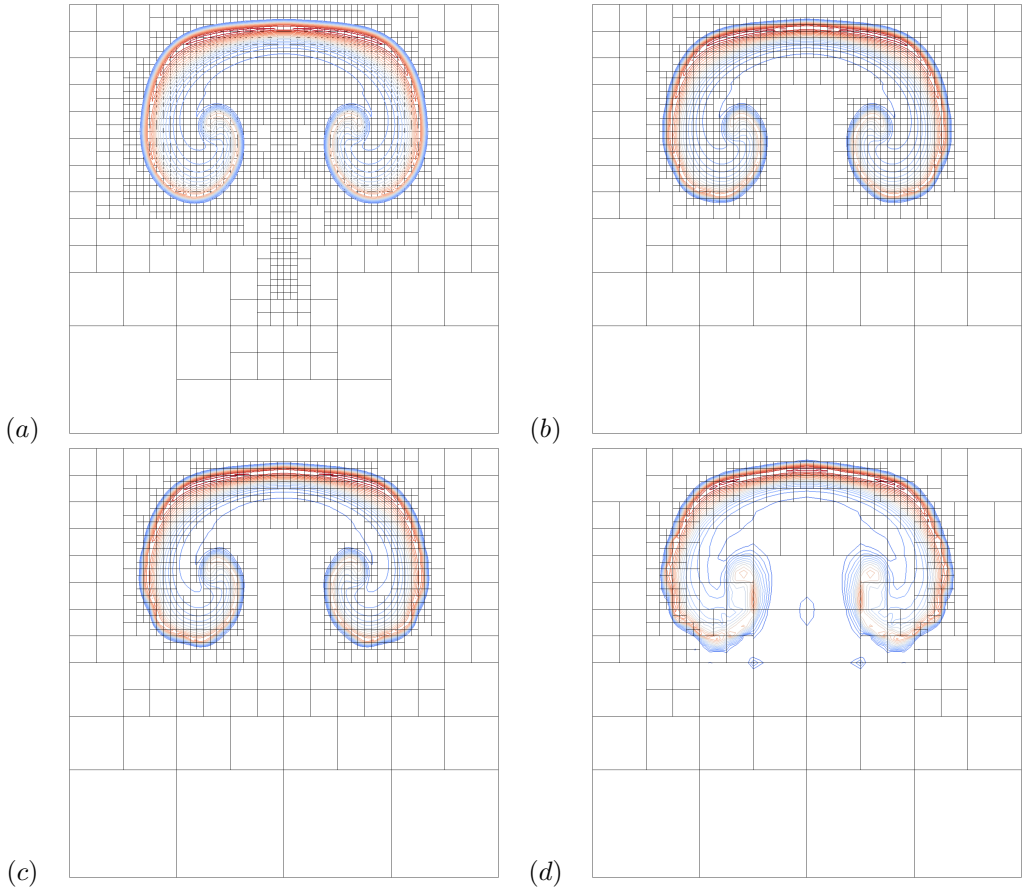

Figure 18: Potential temperature perturbation contours at $t = 700$s for (a) $\theta_t = 0.001$, (b) $\theta_t = 0.01$, (c) $\theta_t = 0.1$, and (d) $\theta_t = 0.35$.

the reference solution almost perfectly, but obtains this result in a significantly shorter simulation time (25%). In this case IMEX methods run with a significantly higher Courant number than in Case 1, which causes the reference simulations for both BDF2 and ARK2 to differ slightly from the explicit one. Similarly, the AMR results for both ARK2 and BDF2 are not exactly the same as their explicit counterparts. The ARK2 method proves to be the fastest among the three.

In Fig. 19 the L_2 norms of the potential temperature perturbation for two IMEX methods are presented. The norms were computed using the reference simulation for each method (BDF2 AMR simulations compared with BDF2 reference, likewise for ARK2). Similarly to the Case 1 results, the error does not differ much among methods, which indicates that AMR affects the accuracy of both methods in the same way. The level of error is significantly higher, though, than in the density current case. This is because the artificial viscosity applied to stabilize this flow is significantly smaller than for the density current case. Typically the viscosity for Case 1 is $\mu = 75\text{m}^2/\text{s}$ while for Case 2 it is $\mu = 0.1\text{m}^2/\text{s}$. This amount of viscosity is enough to stabilize the flow, but is barely enough to guarantee a smooth solution for the effective resolution of $\Delta x = 3.125$m (see [32]). A more in-depth analysis of the rising thermal bubble case with AMR will follow in an upcoming paper.

8.2.2. Performance analysis

Figure 20 shows the simulation runtime (Fig. 20a) and speed-up (Fig. 20b) of AMR simulations for different refinement thresholds. Solid lines represent unoptimized simulations, while dashed lines show the

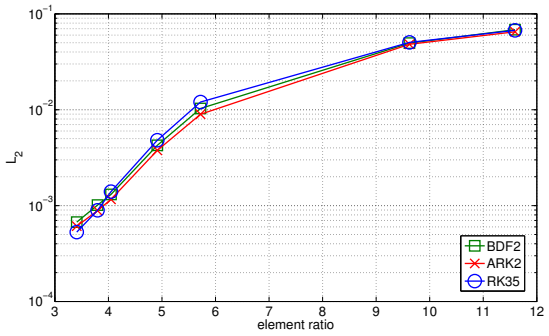

Figure 19: L_2 error norms for RK35 (blue line with circles), BDF2 (green line with squares) and ARK2 (red line with crosses) for Case 2.

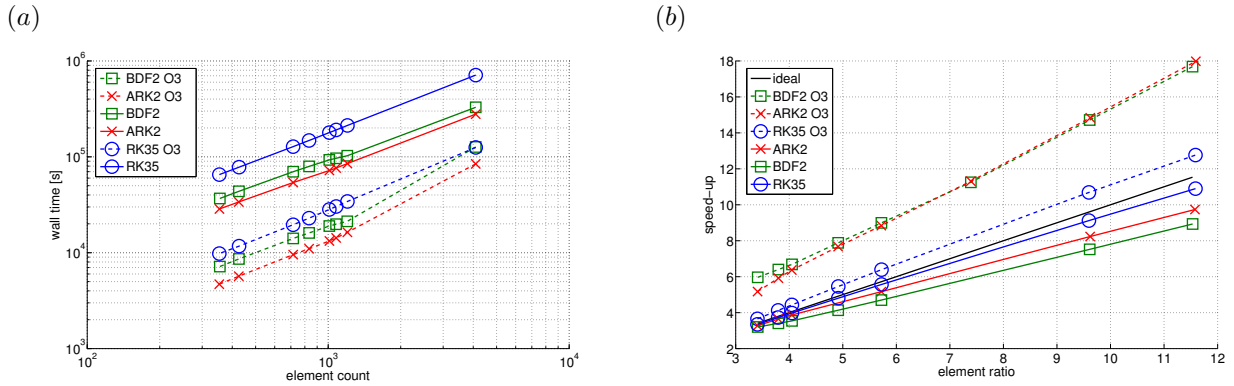

Figure 20: (a) Wall clock time as a function of average element count and (b) speed-up of AMR simulations as a function of ER for different IMEX methods for Case 2. Green line with squares represents BDF2; red line with crosses shows the result for ARK2; blue line with circles marks the timing of the explicit RK35 method. Dashed lines represent the code optimized using compiler flags, solid lines show the unoptimized case. The solid black line shows the expected ideal speed-up.

performance of the compiler O3 optimizations. As observed already in Table III, ARK2 was the fastest one for this case for both optimized and unoptimized runs. The speed-up plot for unoptimized simulations looks similar to those for Case 1, where explicit RK35 obtains nearly ideal speed-up, followed by ARK2 and BDF2. This time the difference between IMEX methods is not as pronounced. The compiler optimized runs again show speed-ups exceeding expectations.

The greater speed of ARK2 simulations can be credited to an increased Courant number (compared to Case 1) and the absence of preconditioning. ARK2 is a two stage method, where each of the stages performs significantly less GMRES iterations than a single BDF2 time-step. The total number of iterations per time step is presented in Fig. 21. Since the cost of GMRES is $O(k^2)$ (where k is the number of GMRES iterations) that alone accounts for an increased runtime. On the other hand, the use of preconditioners could significantly bring down the number of iterations for BDF2, while ARK2 may not benefit as much due to the relatively small iteration count at each stage. We discuss the impact of preconditioners in the following section.

Figure 21 shows the average number of GMRES iterations for both IMEX methods. For ARK2 we present the sum of iterations from both implicit stages. In both cases AMR causes the increase of the iteration count, however it is much more pronounced for the BDF2 method. Contrary to the Case 1 result, this does not translate into a significant difference in the speed-up plots.

Looking at Fig. 22 we see that the previously reported spikes in the GMRES iteration count (for Case

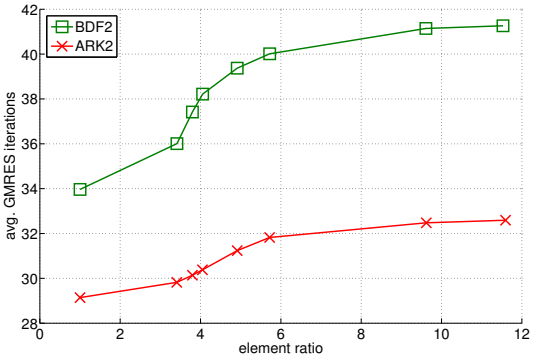

Figure 21: Average number of GMRES iterations as a function of ER for different IMEX methods for Case 2. Green line with squares represents BDF2; red line with crosses shows the result for ARK2.

1) are not as significant for Case 2. The iteration count for the BDF2 AMR simulation (bottom plot of Fig. 22a) is larger compared to the reference case (top plot). The spikes, however, mainly occur while de-refining the mesh (Fig. 22b) and do not significantly contribute to the overall average iteration count. Very small spikes can be noted when the element count increases, but compared to the average iteration number this increase is negligible. For ARK2 the AMR GMRES iteration time history (bottom plot of Fig. 22c) much more closely follows the reference case (top plot of Fig. 22c), which indicates more robustness of this method with respect to the AMR algorithm.

8.2.3. Preconditioning

In any real application IMEX methods are used in combination with a preconditioner in order to improve the conditioning of the system and bring the GMRES iteration count down. Of course, the cost of computing and applying the preconditioner has to be smaller than the cost saved by decreasing the iteration count. An added complexity occurs in the AMR case, because the mesh keeps changing and therefore the preconditioner has to adapt to those changes. Here we present only a very preliminary discussion of the use of preconditioners in the AMR simulations. We apply the element-based spectrally-optimized approximate inverse preconditioner described in [41].

Figure 23 presents the GMRES iteration count for both IMEX methods with preconditioning (dashed line) and without (solid line). For both BDF2 and ARK2, the preconditioner brought down the number of iterations by a similar factor. The decrease of the iteration count does not translate to decrease in runtime for both methods equally well, which can be observed in Fig. 24a. For BDF2 the use of preconditioning resulted in noticeably faster simulations, while for ARK2 the speed-up was not as pronounced. Interestingly, the reference simulations benefitted more from the preconditioner (the rightmost points on the graph). This is due to the fact that without AMR the preconditioner has to be computed only once at the beginning of the simulation and does not need to be recomputed. For AMR simulations the recomputing of the preconditioner adds a significant overhead. It is apparent in Fig. 24b, where the speed-up lines for preconditioned simulations are significantly lower than the speed-up for non-preconditioned cases.

It is worth noting that this is just a preliminary study of the influence of preconditioning on the AMR simulations. At the moment we recompute the preconditioner each time the mesh is modified, even if only one of the elements is changed. Further study will reveal if it is possible to limit this overhead. Additionally, this preconditioner was designed for the Schur-form CG method and is applied to no-Schur form DG method, therefore it may not be as beneficial for the cases at hand. We will address these issues in an upcoming paper.

8.2.4. Mass conservation

Similarly to Case 1, the mass conservation error for the AMR simulations for Case 2 is bounded by the error for the reference case. Figure 25 shows the same sensitivity of M to an increasing number of elements

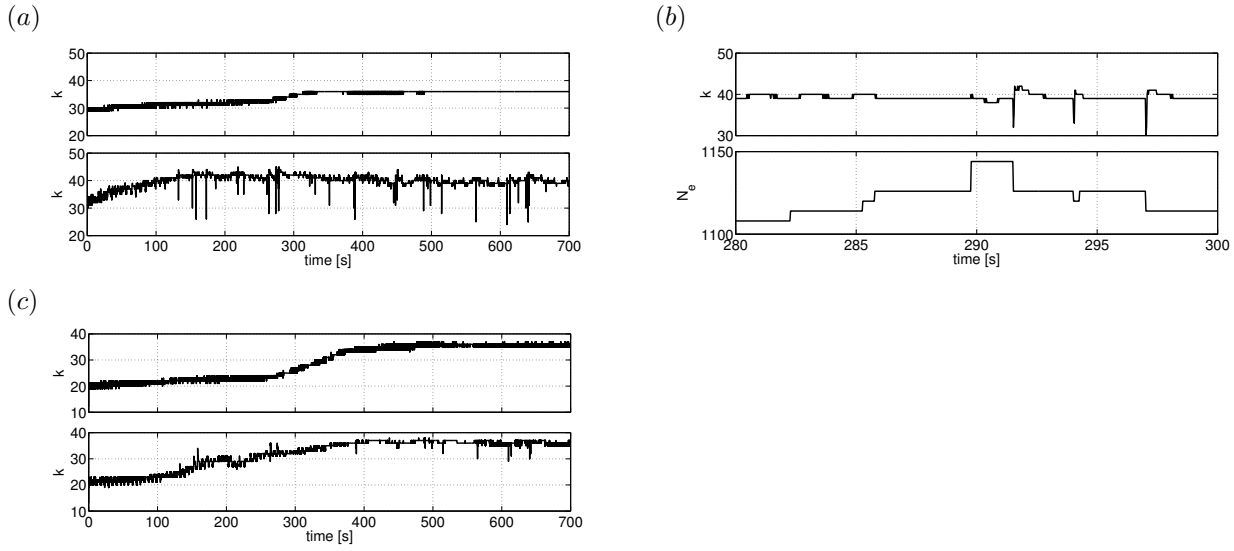

Figure 22: Time history of GMRES iterations (k) for reference (top figure on each panel) and $\theta_t = 0.1$ (bottom figure on each panel) simulations for (a) BDF2 and (c) ARK2 simulation. Panel (b) contains the close-up of the time history of k between $t = 280$s and 300s for BDF2 simulation (top figure) and the time history of the element count (N_e on the bottom figure).

Table IV: Corrected coefficients for the RK35 method. * marks the difference from [42].

stage	α			β
1	1	0	0	0.377268915331368
2	0	1	0	0.377268915331368
3	0.355909775063326*	0.644090224936674	0	0.242995220537396
4	0.367933791638137	0.632066208361863	0	0.238458932846290
5	0	0.762406163401431	0.237593836598569	0.237593836598569

over simulation time. All three time integration methods give consistent mass conservation behavior. During the study an important feature of the explicit RK35 method has been noted, which can be extended to other time integration methods. It is important that the coefficients of each stage of the Runge-Kutta method are consistent up to a desired precision. If we aim for double precision and therefore are looking for round-off errors at the 10^{-16} level, the coefficients have to be consistent up to the 16^{th} decimal place. This is not the case with the coefficients of the third stage of RK35 published in Ruuth [42]. In the case of this study a small inconsistency at the last decimal place, which is probably the result of a roundoff error, led to a steady growth of the M norm. In Table IV we provide consistent coefficients for the RK35 method which fixed this steady growth.

8.2.5. AMR cost breakdown

Compared to the results for Case 1, the share of the cost of AMR for IMEX methods in Case 2 is significantly smaller. This is because the time integration became more expensive due to the increased Courant number. Also, the criterion evaluation cost with regards to the total AMR cost is much bigger than in the previous case, especially for $\theta_t = 0.01$ simulations. This is because we are checking the refinement criterion every 10 iterations, which translates to 0.25s of simulation time. We therefore check for refinement more often than actually perform any mesh modification. It is clear that the refinement check frequency parameter could have been chosen more optimally. Note, however, that for this case the cost of time integration is so high that even more frequent checks would not cause the cost of AMR to exceed a few percents.

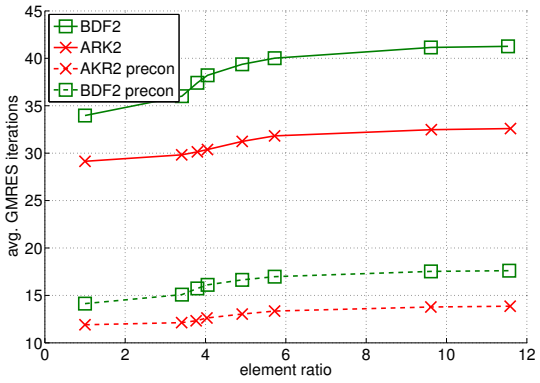

Figure 23: Average number of GMRES iterations as a function of ER for different IMEX methods for Case 2 with (dashed line) and without preconditioning (solid line). Green line with squares represents BDF2; red line with crosses shows the result for ARK2.

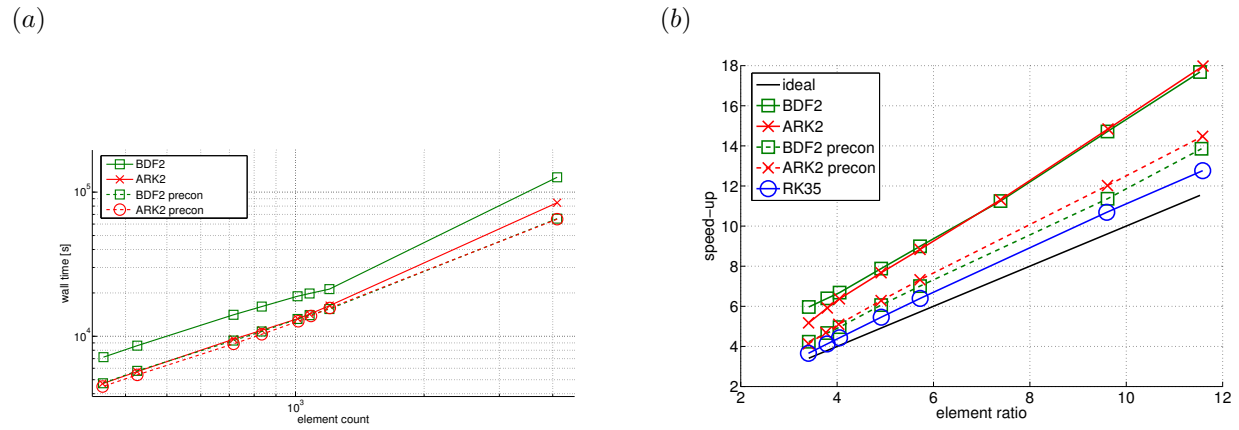

Figure 24: (a) Runtime and (b) speed-up of compiler optimized IMEX simulations for Case 2 with (dashed line) and without preconditioning (solid line). Green line with squares represents BDF2; red line with crosses shows the result for ARK2.

9. Conclusion

We presented the details of the AMR algorithm for the discontinuous Galerkin version of the Non-hydrostatic Unified Model of the Atmosphere (NUMA) and sought to analyze the benefit of AMR. To this end, we compared the accuracy versus the speed-up of AMR with respect to uniform grid simulations. The AMR algorithm was tested using a threshold based criterion for both the density current and rising thermal bubble cases. We have investigated the performance of the AMR algorithm running with three time integration methods: explicit RK35 and two IMEX methods: BDF2 and ARK2. For all the methods the accuracy of the result depends on the choice of the threshold for the refinement criterion. For higher element ratios (ER) the criterion does not capture all the features of the solution, therefore the L2 error increases from 10^{-6} for low threshold simulations to 10^{-4} for high threshold ones for the density current case. The most significant features of the solution such as the front position, minimum potential temperature or the general structure of the front are captured correctly even for high ER simulations. The ARK2 method reproduces closely the result of RK35, while BDF2 provides a less similar agreement.

For the rising thermal bubble case the general features of the solution are captured correctly by all the simulations except the one with the highest threshold value. In that case the boundary of the bubble is

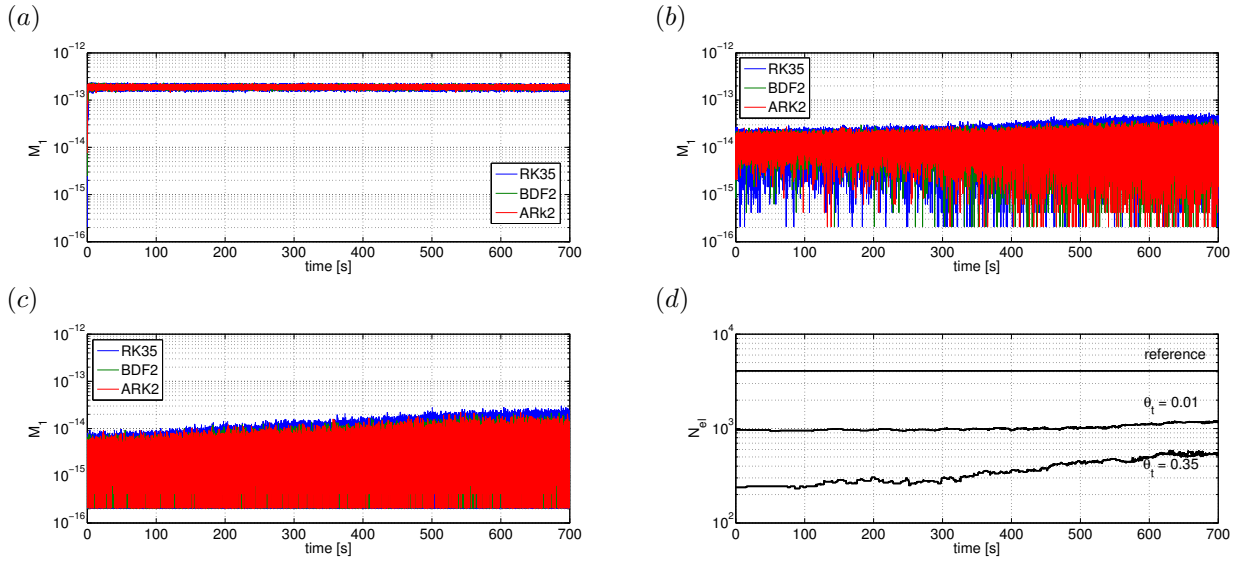

Figure 25: Mass conservation error for Case 2 for (a) reference simulation, (b) $\theta_t = 0.01$ AMR simulation and (c) $\theta_t = 0.35$ AMR simulation for three time integration methods. Panel (d) shows the time history of the element count for all three simulations using all three methods. Results from different simulations with the same time-integration method overlap, which is expected.

Table V: Timings (in seconds of runtime) and percentage breakdown of different AMR components for Case 2

	RK35		BDF2		ARK2	
	$\theta_t = 0.001$	$\theta_t = 4.0$	$\theta_t = 0.001$	$\theta_t = 4.0$	$\theta_t = 0.001$	$\theta_t = 4.0$
total	27994 (100%)	9710 (100%)	18938 (100%)	7154 (100%)	13245 (100%)	4687 (100%)
volume integrals	13954 (49.8%)	4740 (48.8%)	4990 (26.3%)	1829 (25.6%)	4618 (34.9%)	1655 (35.3%)
face integrals	2560 (9.14%)	678.9 (7%)	1540 (8.13%)	385.7 (5.39%)	1369 (10.3%)	342.6 (7.31%)
non-conforming faces	677.7 (2.42%)	503.6 (5.19%)	535.4 (2.83%)	440.6 (6.16%)	460.3 (3.47%)	374.4 (7.99%)
AMR:	21.7 (0.08%)	8.5 (0.09%)	22.2 (0.12 %)	8.49 (0.11%)	21.8 (0.16%)	8.29 (0.18%)
criterion evaluation	16.8	5.9	16.8	5.9	16.8	5.93
mesh manipulation	0.15	0.16	0.15	0.16	0.15	0.15
data projection	1.76	0.86	2.39	1.13	1.93	0.9
other	2.5	1.11	2.7	1.16	2.81	1.22

clearly disturbed and some mesh artifacts are present in the solution. It was discovered that not only the refinement in the high temperature gradient area at the boundary of the bubble needs to be resolved, but the resolution of the interior of the bubble plays a role. All the important features such as the position of the bubble and maximum potential temperature were captured very accurately by the lower refinement thresholds at a fraction of the cost.

The performance of the algorithm was investigated by comparing the runtime and speed-up of different AMR simulations. The explicit RK35 showed nearly perfect speed-up which confirms that the AMR algorithm is indeed very efficient. In the cases shown in the paper the overall cost of the adaptive mesh algorithm is below 1% of the total runtime. The main component of the AMR overhead is the evaluation of the refinement criterion, while the mesh manipulation and data projections have negligible cost. For the density current test case (Case 1) with BDF2 an additional big overhead was the increased GMRES iteration count caused by AMR. The BDF2 speed-up turned out to be the worst of all the three methods. Both IMEX methods had runtimes about an order of magnitude smaller than RK35. ARK2 was shown to be slightly slower than BDF2 using the same time step, but with much increased accuracy and improved speed-up properties. The ARK2 method was shown to be more robust (less sensitive) to the changing mesh caused by the AMR algorithm. For the rising thermal bubble test case (case 2), the ARK2 was the fastest

method due to the increased Courant number. The difference in speed-up curves was not as pronounced, but again the ARK2 proved to have better properties and less overhead than BDF2.

In nonhydrostatic atmospheric applications (when using the fully compressible equations), IMEX methods are an absolute necessity, due to a very strict constraint imposed on the explicit time-step by acoustic waves. This study shows that the performance of some IMEX methods can be affected by a dynamically adaptive mesh. It is important not only to design the AMR algorithm in an efficient way, but also to consider which time integration method to use in such applications. In our future investigations we will pursue the ARK2 method due to its good performance properties and excellent accuracy.

The mass conservation error for both test cases and all the time integration methods was shown to be bounded by the mass conservation error of the reference simulations. For adaptive simulations the mass conservation was affected by the changing number of elements in the mesh.

A preliminary study of the influence of preconditioning on the AMR simulations showed that it is another significant source of overhead, mainly because the preconditioner was recomputed after every mesh modification. We will pursue the possibilities of reducing this overhead in future work since it may not be necessary to always recompute the preconditioner.

Finally, the effect of the compiler optimizations on the algorithm performance was investigated. The optimizations provide significant runtime reduction of the simulations, as big as a factor of 10. Additionally, optimization affects the speed-up of the AMR algorithm, as more efficient memory handling for smaller problems allows for speed-ups greater than it would follow from the simple performance model. In a sense the optimizations more than make up for the overhead caused by the AMR.

Acknowledgements

The authors gratefully acknowledge the support of the Office of Naval Research through program element PE-0602435N, the National Science Foundation (Division of Mathematical Sciences) through program element 121670, and the Air Force Office of Scientific Research through the Computational Mathematics program. The authors are grateful to Andreas Müller and Simone Marras for constructive discussions that greatly contributed to this paper.

[1] R. D. Nair, H.-W. Choi, H. M. Tufo, Computational aspects of a scalable high-order discontinuous Galerkin atmospheric dynamical core, Comp. Fl. 38 (2) (2009) 309–319.

[2] J. F. Kelly, F. X. Giraldo, Continuous and discontinuous Galerkin methods for a scalable 3D nonhydrostatic atmospheric model: limited area mode, J. Comp. Phys. 231 (2) (2012) 7988–8008.

[3] R. Hartmann, P. Houston, Adaptive discontinuous Galerkin finite element methods for the compressible Euler equations, J. Comp. Phys. 183 (2) (2002) 508–532.

[4] P. Bastian, M. Blatt, A. Dedner, C. Engwer, R. Klöfkorn, R. Kornhuber, M. Ohlberger, O. Sander, A generic grid interface for parallel and adaptive scientific computing. Part II: Implementation and tests in DUNE, Computing 82 (2-3) (2008) 121–138.

[5] C. Burstedde, O. Ghattas, M. Gurnis, G. Stadler, E. Tan, T. Tu, L. C. Wilcox, S. Zhong, Scalable adaptive mantle convection simulation on petascale supercomputers, in: Proc. 2008 ACM/IEEE Supercomputing, IEEE Press, 2008, p. 62.

[6] C. Jablonowski, Adaptive grids in weather and climate modeling, Ph.D. thesis, The University of Michigan (2004).

[7] J. Behrens, Adaptive atmospheric modeling: key techniques in grid generation, data structures, and numerical operations with applications, Springer, 2006.

[8] G. W. Ley, R. L. Elsberry, Forecasts of typhoon Irma using a nested grid model, Mon. Wea. Rev. 104 (1976) 1154. doi:10.1175/1520-0493(1976)104¡1154:FOTIUA¿2.0.CO;2.

[9] Y. Kurihara, M. A. Bender, Use of a movable nested-mesh model for tracking a small vortex, Mon. Wea. Rev. 108 (1980) 1792–1809. doi:10.1175/1520-0493(1980)108¡1792:UOAMNM¿2.0.CO;2.

[10] D.-L. Zhang, H.-R. Chang, N. L. Seaman, T. T. Warner, J. M. Fritsch, A two-way interactive nesting procedure with variable terrain resolution, Mon. Wea. Rev. 114 (7) (1986) 1330–1339.

[11] G. S. Dietachmayer, K. K. Droegemeier, Application of continuous dynamic grid adaptation techniques to meteorological modeling. I: Basic formulation and accuracy, Mon. Wea. Rev. 120 (8) (1992) 1675–1706.

[12] J. M. Prusa, P. K. Smolarkiewicz, An all-scale anelastic model for geophysical flows: dynamic grid deformation, J. Comp. Phys. 190 (2) (2003) 601–622.

[13] C. J. Budd, W. Huang, R. D. Russell, Adaptivity with moving grids, Acta Numerica 18 (1) (2009) 111–241.

[14] W. C. Skamarock, J. Oliger, R. L. Street, Adaptive grid refinement for numerical weather prediction, J. Comp. Phys. 80 (1) (1989) 27–60.

[15] W. C. Skamarock, J. B. Klemp, Adaptive grid refinement for two-dimensional and three-dimensional non-hydrostatic atmospheric flow, Mon. Wea. Rev. 121 (3) (1993) 788–804.

[16] M. J. Berger, J. Oliger, Adaptive mesh refinement for hyperbolic partial differential equations, J. Comp. Phys. 53 (3) (1984) 484–512.
[17] M. J. Berger, P. Colella, Local adaptive mesh refinement for shock hydrodynamics, J. Comp. Phys. 82 (1) (1989) 64–84.
[18] R. J. LeVeque, Wave propagation algorithms for multidimensional hyperbolic systems, J. Comp. Phys. 131 (2) (1997) 327–353.
[19] N. Nikiforakis, AMR for global atmospheric modelling, in: Adaptive Mesh Refinement-Theory and Applications, Springer, 2005, pp. 505–526.
[20] D. P. Bacon, N. N. Ahmad, Z. Boybeyi, T. J. Dunn, M. S. Hall, P. C. S. Lee, R. A. Sarma, M. D. Turner, K. T. Waight III, S. H. Young, et al., A dynamically adapting weather and dispersion model: the operational multiscale environment model with grid adaptivity (OMEGA), Mon. Wea. Rev. 128 (7) (2000) 2044–2076.
[21] P. K. Smolarkiewicz, A fully multidimensional positive definite advection transport algorithm with small implicit diffusion, J. Comp. Phys. 54 (2) (1984) 325–362.
[22] J. P. Iselin, J. M. Prusa, W. J. Gutowski, Dynamic grid adaptation using the MPDATA scheme, Mon. Wea. Rev. 130 (4) (2002) 1026–1039.
[23] F. X. Giraldo, The Lagrange-Galerkin method for the two-dimensional shallow water equations on adaptive grids, Int. J. Numer. Meth. Fl. 33 (6) (2000) 789–832.
[24] J. Behrens, Atmospheric and ocean modeling with an adaptive finite element solver for the shallow-water equations, App. Num. Math. 26 (1) (1998) 217–226.
[25] J. Behrens, N. Rakowsky, W. Hiller, D. Handorf, M. Läuter, J. Päpke, K. Dethloff, amatos: Parallel adaptive mesh generator for atmospheric and oceanic simulation, Ocean Mod. 10 (1) (2005) 171–183.
[26] M. A. Taylor, A. Fournier, A compatible and conservative spectral element method on unstructured grids, J. Comp. Phys. 229 (17) (2010) 5879–5895.
[27] A. St-Cyr, C. Jablonowski, J. M. Dennis, H. M. Tufo, S. J. Thomas, A comparison of two shallow water models with non-conforming adaptive grids: classical tests, arXiv preprint physics/0702133.
[28] E. J. Kubatko, S. Bunya, C. Dawson, J. J. Westerink, Dynamic¡ i¿ p¡/i¿-adaptive Runge–Kutta discontinuous Galerkin methods for the shallow water equations, Comp. Meth. Appl. Mech. Engng. 198 (21) (2009) 1766–1774.
[29] D. A. Kopriva, A conservatice staggered-grid chebyshev multidomain method for compressible flows. II: A semi-stuctured method, Tech. Rep. 2, NASA Contractor Report (Oct. 1996). doi:10.1006/jcph.1996.0225.
URL http://linkinghub.elsevier.com/retrieve/pii/S0021999196902259
[30] Y. Maday, C. Mavriplis, A. T. Patera, Nonconforming mortar element methods: Application to spectral discretizations, Institute for Computer Applications in Science and Engineering, NASA Langley Research Center, 1988.
[31] D. Rosenberg, A. Fournier, P. Fischer, A. Pouquet, Geophysical–astrophysical spectral-element adaptive refinement (GASpAR): Object-oriented h-adaptive fluid dynamics simulation, J. Comp. Phys. 215 (1) (2006) 59–80.
[32] A. Mueller, J. Behrens, F. X. Giraldo, V. Wirth, An adaptive discontinuous Galerkin method for modeling atmospheric convection, J. Comp. Phys. 235 (1) (2012) 371–393.
[33] S. Brdar, M. Baldauf, A. Dedner, R. Klöfkorn, Comparison of dynamical cores for NWP models: comparison of COSMO and Dune, Theor. Comp. Fl. Dyn. (2012) 1–20.
[34] C. Eskilsson, An hp-adaptive discontinuous Galerkin method for shallow water flows, Int. J. Numer. Meth. Fl. 67 (11) (2011) 1605–1623.
[35] S. Blaise, A. St-Cyr, A dynamic hp-adaptive discontinuous Galerkin method for shallow-water flows on the sphere with application to a global tsunami simulation, Mon. Wea. Rev. 140 (3) (2012) 978–996.
[36] F. X. Giraldo, M. Restelli, A study of spectral element and discontinuous Galerkin methods for the Navier-Stokes equations in non-hydrostatic mesoscale atmospheric modeling: equation sets and test cases, J. Comp. Phys. 227 (1) (2008) 3849–3877.
[37] C. Burstedde, O. Ghattas, M. Gurnis, T. Isaac, G. Stadler, T. Warburton, L. C. Wilcox, Extreme-scale AMR, Proc. 2010 ACM/IEEE Int. Confence for High Performance Computing, Networking, Storage and Analysis (1) (2010) 1–12.
[38] H. Sundar, R. S. Sampath, G. Biros, Bottom-up construction and 2:1 balance refinement of linear octrees in parallel, SIAM J. Sci. Comp. 30 (5) (2008) 2675–2708. doi:10.1137/070681727.
URL http://epubs.siam.org/doi/abs/10.1137/070681727
[39] J. M. Straka, R. B. Wilhelmson, J. R. Anderson, K. K. Droegemeier, Numerical solutions of a non-linear density current: a benchmark solution and comparisons, Int. J. Numer. Meth. Fl. 17 (1993) 1–22.
[40] F. X. Giraldo, J. F. Kelly, E. M. Constantinescu, Implicit-explicit formulations for a 3D Nonhydrostatic Unified Model of the Atmosphere (NUMA), SIAM J. Sci. Comp. (1).
[41] L. E. Carr, C. F. Borges, F. X. Giraldo, An element-based spectrally-optimized approximate inverse preconditioner for the Euler equations, SIAM J. Sci. Comp. 34 (2012) B392–B420.
[42] S. J. Ruuth, Global optimization of explicit strong-stability-preserving Runge-Kutta methods, Math. Comp. 75 (253) (2006) 183–207.

Appendix A

1D projection

The integral projection method presented in this section was proposed by [29] for general h-p non-conforming elements. Here we describe the method applied to a specific h-non-conforming edge in 2:1 balance.

Let $\xi \in [-1,1]$ denote the coordinate in the standard element space corresponding to the parent (left) side of the edge. Define $z^{(1)}, z^{(2)} \in [-1,1]$ as the coordinates of standard elements corresponding to two children (right) sides of the edge. Let

$$z^{(1)} = \frac{\xi - o^{(1)}}{s}, \qquad z^{(2)} = \frac{\xi - o^{(2)}}{s} \tag{21}$$

be a map $\xi \to z^{(k)}, k = 1,2$ from the parent space to children spaces, where $o^{(k)}$ is the offset parameter for the child k and s is the scale parameter. In our case $o^{(1)} = -0.5$, $o^{(2)} = 0.5$ and $s = 0.5$. The inverse mapping $z^{(k)} \to \xi$ is

$$\xi = s \cdot z^{(k)} + o^{(k)}, \quad k = 1,2. \tag{22}$$

We can now expand the variables using a polynomial basis as follows:

$$q^L(\xi) = \sum_{j=0}^{N} q_j^L \psi_j(\xi), \tag{23}$$

$$q^{Lk}(z^{(k)}) = \sum_{j=0}^{N} q_j^{Lk} \psi_j(z^{(k)}), \quad k = 1,2. \tag{24}$$

By substituting (22) into (23) we get

$$q^L(z^{(k)}) = \sum_{j=0}^{N} q_j^L \psi_j(s \cdot z^{(k)} + o^{(k)}), \quad k = 1,2. \tag{25}$$

In order to perform projection from a parent side to two children sides of the edge we require that for both children sides

$$\int_{-1}^{1} \left(q^{Lk}(z^{(k)}) - q^L(z^{(k)}) \right) \psi_i(z^{(k)}) dz^{(k)} = 0, \quad k = 1,2. \tag{26}$$

Substitution of (24) and (25) to (26) and rearranging yields

$$\sum_{j=0}^{N} \left(\int_{-1}^{1} \psi_j(z^{(k)}) \psi_i(z^{(k)}) dz^{(k)} \right) q_j^{Lk} - \sum_{j=0}^{N} \left(\int_{-1}^{1} \psi_j(s \cdot z^{(k)} + o^{(k)}) \psi_i(z^{(k)}) dz^{(k)} \right) q_j^L = 0. \tag{27}$$

Since $z^{(k)} \in [-1,1]$ regardless of k, we can write $z = z^{(k)}$, which simplifies the notation. The terms in brackets can be represented in matrix form as

$$\mathbf{M}_{ij} = \int_{-1}^{1} \psi_i(z) \psi_j(z) dz, \qquad \mathbf{S}_{ij}^{(k)} = \int_{-1}^{1} \psi_i(z) \psi_j(s \cdot z + o^{(k)}) dz, \quad k = 1,2, \tag{28}$$

which simplifies equation (27) to

$$\mathbf{M}_{ij} q_j^{Lk} - \mathbf{S}_{ij}^{(k)} q_j^L = 0, \quad k = 1,2. \tag{29}$$

Note that $\mathbf{M}_{ij}$ is the standard 1D mass matrix, which is easily invertible. If $\mathbf{P}_{ij}^{Sk} = \mathbf{M}_{ij}^{-1} \mathbf{S}_{ij}^{(k)}$, then

$$q_i^{Lk} = \mathbf{P}_{ij}^{Sk} q_j^L, \quad k = 1,2 \tag{30}$$

and we call $\mathbf{P}^{Sk}$ the scatter projection matrix.

Similarly the gather projection from two children sides to a parent side is performed (see Fig. 5b). We require that on the parent side of the face

$$\int_{-1}^{1} \left(q^R(\xi) - \tilde{q}^R(\xi)\right) \psi_i(\xi) d\xi = 0, \tag{31}$$

where q^R is the continuous projection of the variables q^{R1} and q^{R2} from children sides to a parent side, and

$$\tilde{q}^R(\xi) = \begin{cases} q^{R1}(z^{(1)}) = q^{R1}\left(\frac{\xi - o^{(1)}}{s}\right) & for \quad -1 \le \xi \le 0^-, \\ q^{R2}(z^{(2)}) = q^{R2}\left(\frac{\xi - o^{(2)}}{s}\right) & for \quad 0^+ \le \xi \le 1. \end{cases} \tag{32}$$

Note that $\tilde{q}^R(\xi)$ allows for a discontinuity at $\xi = 0$. Substituting (32) into (31) yields

$$\int_{-1}^{0} \left(q^R(\xi) - q^{R1}\left(\frac{\xi - o^{(1)}}{s}\right)\right) \psi_i(\xi) d\xi + \int_{0}^{1} \left(q^R(\xi) - q^{R2}\left(\frac{\xi - o^{(2)}}{s}\right)\right) \psi_i(\xi) d\xi = 0,$$

Using an expansion analogous to (24), (25) and rearranging we get

$$\left(\int_{-1}^{1} \psi_i(\xi)\psi_j(\xi) d\xi\right) q_j^R - \left(\int_{-1}^{0} \psi_i(\xi)\psi_j\left(\frac{\xi - o^{(1)}}{s}\right) d\xi\right) q_j^{R1} - \left(\int_{0}^{1} \psi_i(\xi)\psi_j\left(\frac{\xi - o^{(2)}}{s}\right) d\xi\right) q_j^{R2} = 0.$$

We introduce the variable change $\xi = s \cdot z + o^{(k)}, \quad d\xi = s \cdot dz$ to the second and third integral, that gives

$$\left(\int_{-1}^{1} \psi_i(\xi)\psi_j(\xi) d\xi\right) q_j^R - s \sum_{k=1}^{2} \left(\int_{-1}^{1} \psi_i(s \cdot z + o^{(k)})\psi_j(z) dz\right) q_j^{Rk} = 0. \tag{33}$$

Note that the term in brackets to the left of q_j^{Rk} is the transpose of $S_{ij}^{(k)}$ defined in (**??**). We can write the integrals in matrix notation:

$$\mathbf{M}_{ij} q_j^R - s \sum_{k=1}^{2} \mathbf{S}_{ij}^{(k)^T} q_j^{Rk} = 0, \tag{34}$$

where $\mathbf{M}_{ij}$ is the mass matrix as defined in (28).Finally, if we define $\mathbf{P}_{ij}^{Gk} = s \cdot \mathbf{M}_{ij}^{-1} \mathbf{S}_{lj}^{(k)^T}$ to be the gather projection matrix, then

$$q_i^R = \sum_{k=1}^{2} \mathbf{P}_{ij}^{Gk} q_j^{Rk}. \tag{35}$$

2D projection

The 2D projection presented here is a two-dimensional extension of the integral projection method presented in the previous section.

Figure 6 presents the standard element corresponding to the parent element with coordinates $(\xi, \eta) \in [-1, 1]^2$ and four children, each with separate coordinates $(z_1^{(k)}, z_2^{(k)}) \in [-1, 1]^2$, where $k = 1, \ldots, 4$. We define a map:

$$\begin{aligned} z_1^{(k)} &= \frac{\xi - o_1^{(k)}}{s}, \\ z_2^{(k)} &= \frac{\eta - o_2^{(k)}}{s}, \end{aligned} \tag{36}$$

where $o_1^{(k)}$ and $o_2^{(k)}$ are offset parameters corresponding to each child element k and coordinate z_1 and z_2. The inverse mapping is now

$$\xi = s \cdot z_1^{(k)} + o_1^{(k)}, \tag{37}$$

$$\eta = s \cdot z_2^{(k)} + o_2^{(k)}, \tag{38}$$

where $k = 1, \ldots, 4$, the scale parameter $s = 0.5$ and offsets $o_i^{(k)} = \pm 0.5$ depending on direction i and element number k.

Each element has a polynomial basis defined in which we expand the projected variable,

$$q^P(\xi, \eta) = \sum_{j=1}^{M_N} q_j^P \psi_j(\xi, \eta), \tag{39}$$

$$q^{Ck}(z_1^{(k)}, z_2^{(k)}) = \sum_{j=1}^{M_N} q_j^{Ck} \psi_j(z_1^{(k)}, z_2^{(k)}), \tag{40}$$

where q^P is the parent element variable and q^{Ck} is the k-th child element variable projected from q^P, and M_N is the number of nodal points in the element. Similarly as in Equation (25) we can substitute the inverse map (37) to (39) and represent the parent variable in terms of the children coordinate system.

$$q^P(\xi, \eta) = q^P(z_1^{(k)}, z_2^{(k)}) = \sum_{j=1}^{M_N} q_j^P \psi_j(s \cdot z_1^{(k)} + o_1^{(k)}, s \cdot z_2^{(k)} + o_2^{(k)}), \qquad k = 1, \ldots, 4. \tag{41}$$

For each $k = 1, \ldots, 4$ we require that

$$\int_{-1}^{1} \int_{-1}^{1} \left(q^{Ck}(z_1^{(k)}, z_2^{(k)}) - q^P(z_1^{(k)}, z_2^{(k)}) \right) \psi_j(z_1^{(k)}, z_2^{(k)}) dz_1^{(k)} dz_2^{(k)} = 0. \tag{42}$$

Substituting expansions (39), (40), rearranging and employing matrix notation yields

$$\mathbf{M}_{ij} q_j^{Ck} - \mathbf{S}_{ij}^{(k)} q_j^P = 0, \tag{43}$$

where

$$\mathbf{M}_{ij} = \int_{-1}^{1} \int_{-1}^{1} \psi_j(z_1, z_2) \psi_i(z_1, z_2) dz_1 dz_2, \tag{44}$$

$$\mathbf{S}_{ij}^{(k)} = \int_{-1}^{1} \int_{-1}^{1} \psi_j(s \cdot z_1 + o_1^{(k)}, s \cdot z_2 + o_2^{(k)}) \psi_i(z_1, z_2) dz_1 dz_2. \tag{45}$$

The projection matrix is once again constructed by inverting the mass matrix and left-multiplying the inverse with Eq. (43). This yields

$$q_i^{Ck} = (\mathbf{P}_{2D}^{Sk})_{ij} q_j^P, \quad k = 1, \ldots, 4, \tag{46}$$

where

$$\mathbf{P}_{2D}^{Sk} = \mathbf{M}^{-1} \mathbf{S}^{(k)}. \tag{47}$$

Similarly as in the case of the gather projection in the 1D non-conforming edge case, the 2D gather projection matrix is constructed by multiplying the inverse of the mass matrix and transpose of 2D $\mathbf{S}^{(k)}$:

$$\mathbf{P}_{2D}^{Gk} = s \cdot \mathbf{M}^{-1} {\mathbf{S}^{(k)}}^T, \tag{48}$$

which yields:

$$q_i^P = \sum_{k=1}^{4} (\mathbf{P}_{2D}^{Gk})_{ij} q_j^{Ck}. \tag{49}$$

This approach is easily extendable to 3D projections of hexahedral elements since the same tensor-product operations are being applied in 1D, 2D, or 3D.